TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

LASSI VÄISÄNEN

OHJELMISTOPROJEKTIN PROJEKTINHALLINTADATAN VISUALISOINTI

Diplomityö

Tarkastaja: Outi Sievi-Korte
Tarkastaja ja aihe hyväksytty
28. maaliskuuta 2018

TIIVISTELMÄ

LASSI VÄISÄNEN: Ohjelmistoprojektin projektinhallintadatan visualisointi
Tampereen teknillinen yliopisto
Diplomityö, 42 sivua, 10 liitesivua
Toukokuu 2018
Tietotekniikan diplomi-insinöörin tutkinto-ohjelma
Pääaine: Pervasive Systems
Tarkastaja: Outi Sievi-Korte

Avainsanat: visualisointi, projektinhallinta, Scrum, Agilefant

Modernissa ohjelmistokehityksessä suositaan yhä ketterämpiä menetelmiä ja näiden menetelmien tukena käytetään yhä enemmän automatisoituja työkaluja tuotannon apuna. Nämä työkalut luovat ja hallitsevat projektiin liittyvää metadataa, jonka tulkitseminen voi auttaa projektin ja menetelmien evaluointia.

Tässä diplomityössä on pyritty selvittämään projektinhallintadatan visualisoinnin mahdollisuuksia ketterän kehityksen opetuksen ja prosessien arvioinnin tueksi. Työn ohessa on toteutettu tapaustutkimus olemassa olevan visualisointityökalun laajentamiseksi niin, että sillä voidaan visualisoida uudenlaista projektinhallintadataa.

ABSTRACT

LASSI VÄISÄNEN: Visualizing software project's project management data
Tampere University of Technology
Master of Science Thesis, 42 pages, 10 Appendix pages
May 2018
Master's Degree Programme in Information Technology
Major: Pervasive Systems
Examiner: Outi Sievi-Korte

Keywords: visualization, project management, Scrum, Agilefant

Modern software development favors ever more agile methodologies and with these methodologies more and more automatized tools are used to help development. These tools create and store project related metadata which can be examined to help evaluate the project's and its methodologies' effectiveness.

This master's thesis aims to explore the possibilities of visualizing the project management data of a software project to help teach and evaluate agile methods. The thesis contains a case study to expand an existing visualization platform to include a new project management data visualization.

ALKUSANAT

Tämä dokumentti on Tampereen teknilliselle yliopistolle tehty diplomityö. Kiitos tietotekniikan laitokselle aiheesta. Kiitos Jeesus.

Tampereella, 21.5.2018

Lassi Väisänen

SISÄLLYSLUETTELO

LYHENTEET JA MERKINNÄT

Agilefant	Projektinhallintatyökalu
Scrum	Ohjelmistokehityksessä käytettävä projektinhallinan viitekehys
Git	Versionhallintajärjestelmä
Github	Git-versionhallintajärjestelmän käytölle tarkoitettu pilvipalvelu
SQL	Relaatiotietokantahakuissa käytetty kyselykieli
N4S-visu	Datan visualisointityökalu
Node.js	Javascript-viitekehys palvelinohjelmointiin
MongoDB	Dokumenttipohjainen tietokantaohjelma
Http	Hypertext Transfer Protocol, www-ohjelmien käyttämä tiedonsiirto-protokolla
REST	arkkitehtuurimalli ohjelmointirajapintojen toteuttamiseen
JSON	Tiedonsiirrossa käytetty tiedostomuoto
URL	Uniform Resource Identifier, www-sivujen osoitin
Ubuntu	Linux-käyttöjärjestelmä

1. JOHDANTO

Tietokonejärjestelmiä ja -ohjelmistoja tuotettaessa projektien laajuudet voivat vaatia jopa kymmeniä tuhansia henkilötyötunteja yhden tuotteen valmistamiseksi. Tämän takia tuotantoon liittyvät prosessit on tunnettava hyvin ja näiden prosessien parantaminen ja tehokkaana pitäminen on elintärkeää ohjelmistoprojektin tavoitteissa pysymiseksi. Esimerkiksi tuotteen vaatimusmäärittelyjen muutos kesken ohjelmiston tuotannon voi haitata tuotteen laatua, pidentää tuotantoaikaa ja nostaa kuluja merkittävästi. [1] Ohjelmistotuotantoprosessien tehostamiseksi on käytetty lukuisia erilaisia lean-periaatteen menetelmiä ja niin sanottuja ketterän kehityksen malleja, joilla tuotannon mukautuvuutta muutoksiin voidaan parantaa.

Modernin ohjelmistokehityksen sivutuotteena kehityksessä käytettävät prosessit tuottavat valtavia määriä dataa. Esimerkiksi ohjelmistokehitysprosessin eri tehtävien etenemiseen käytettävät tehtävienhallintatyökalut, ohjelmistokoodin versiointiin ja historian tallentamiseen käytettävä versionhallintatyökalu ja erilaiset muut projektinhallintatyökalut tuottavat jatkuvan virran tietoa ohjelmistoprojektin käynnissä ollessa sen tilan etenemisestä. Tämä data on kuitenkin hajautettuna useiden eri työkalujen tuottamana erilaisilla alustoilla ja erilaisissa muodoissa. Projektinhallintatyökalujen datan valjastaminen käyttöön automatisoiduilla välineillä voisi tuoda arvokasta lisätietoa projektin prosessien etenemisestä hyvinkin pienellä vaivannäöllä. [2]

Tämän työn tarkoituksena on selvittää mahdollisuuksia ketterien ohjelmistotuotantomenetelmien tuottaman datan keräämiseen ja esittämiseen ohjelmistotuotantoprosessin etenemisen seuraamisen helpottamiseksi. Näiden prosessien tarkastelun helpottaminen voisi auttaa erityisesti ohjelmistotuotantomenetelmien käytön tehokkuuden arviointia opetustyössä, mutta mahdollisesti myös teollisuudessa prosessien retrospektiivisen analyysin tukena. Työssä pyritäänkin vastaamaan kysymykseen: ”Miten projektinhallintadataa voidaan visualisoida ketterän ohjelmistoprojektin etenemisen tarkkailun ja arvioinnin helpottamiseksi?”.

Diplomityössä on kuvattu erään ketterän ohjelmistokehityksen apuna käytettävän projektinhallintatyökalun tuottaman datan visualisointimahdollisuuksia. Apuna on käytetty jo olemassa olevaa yliopistolla kehitettyä visualisointialustaa, jota on pyritty hyödyntämään mahdollisimman tehokkaasti uudenlaisen syötedatan kanssa. Tarkoituksena on luoda projektinhallintatyökalun datasta visualisaationäkymä, josta projektin etenemisen yleiskuva hahmottuu selkeästi ja sen käyttö helpottaa kurssihenkilökunnan työtä projektinhallintatyökalun ja projektinhallintamenetelmien tehokkaan käytön arvioinnissa.

Luvussa kaksi on käyty läpi moderneja ohjelmistonkehitysprosesseja ja niissä käytettyjä menetelmiä. Erityisesti ohjelmistotuotannossa käytettäviä ketteriä menetelmiä on käsitelty ja näiden näiden menetelmien kanssa käytettyjä työkaluja on esitelty.

Neljännessä luvussa on tapaustutkimuksena esitelty olemassa oleva visualisointiohjelma ja esitetty suunnitelma siihen toteutettavaan lisäosaan uuden projektinhallinta-alustan datan tukemiseksi. Tutkittavan projektinhallinta-alustan toimintoja ja käyttökohteita on avattu ja sen tarjoaman datan muoto ja saatavuus on esitelty.

Viidennessä luvussa on käyty läpi visualisointiohjelman laajennuksen kehitys ja sen vaiheet. Visualisointiohjelman vaatimaa datamallia on tutkittu ja suunniteltu, miten toteutettavan lisämoduulin tulisi toimia. Lisäosan kehityksen vaiheet on kirjattu ylös ja ohjelman tuottamien visualisaatioiden versiot eri kehitysvaiheissa on esitelty.

Luvussa kuusi on pureuduttu jo havaittuun verkkoapplikaation ylläpidon ja jakamisen ongelmaan ja siihen, kuinka tätä voitaisiin helpottaa samalla, kun visualisointiohjelmaa laajennetaan. Erilaisia ratkaisuvaihtoehtoja on vertailtu ja arvioitu, ja ongelman ratkaisemiseksi valittu tapa on kuvattu.

Luvussa seitsemän on käyty läpi saadut tulokset ja arvioitu työn tuottamaa lisäarvoa aiheen parissa työskentelyyn ja toteutettujen työkalujen käytön tuottamaa hyötyä erilaisissa tilanteissa.

Viimeisessä luvussa on tehty yhteenveto havainnoista ja summattu työn tavoitteet, sisältö ja tulosten merkitys.

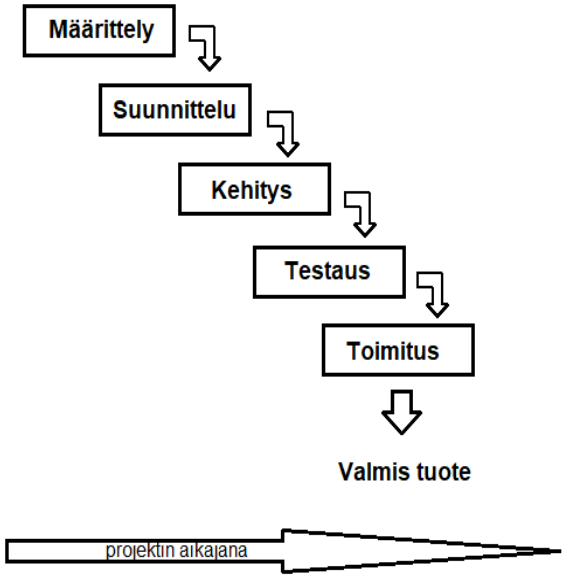
2. OHJELMISTOKEHITYSPROSESSIT

Modernia ohjelmistokehitystä suoritetaan lukuisten erilaisten tietoteknisten työkalujen avulla. Nämä työkalut luovat prosessille datajäljen, joka sisältää valtavan määrän informaatiota ohjelmistoprojektin etenemisestä, sen vaiheista ja tapahtumien tapahtumisajoista. Ohjelmistoprojektin dataa sisältäviä työkaluja voivat olla muun muassa versionhallintajärjestelmä, tehtävienhallintaohjelmisto ja erilaiset jatkuvan integraation alustat. Eri alustojen keräämä data elää kuitenkin hajautettuna näiden alustojen omissa tietokannoissa ja jokaisen alustan datalla on oma muotonsa. Eri lähteiden käyttödatan ristiin arvioiminen ja vertailu voi siis käsin olla työlästä. Onneksi useat pilvipalveluina toimivat ohjelmistotuotantoa helpottavat alustat tarjoavat koneellista käsittelyä varten tiedoilleen ohjelmointirajapinnan, jonka avulla dataa voidaan lukea ja tämän jälkeen jäsentää ja muokata haluttuun muotoon automaattisesti. [3]

2.1 Ketterä ohjelmistokehitys

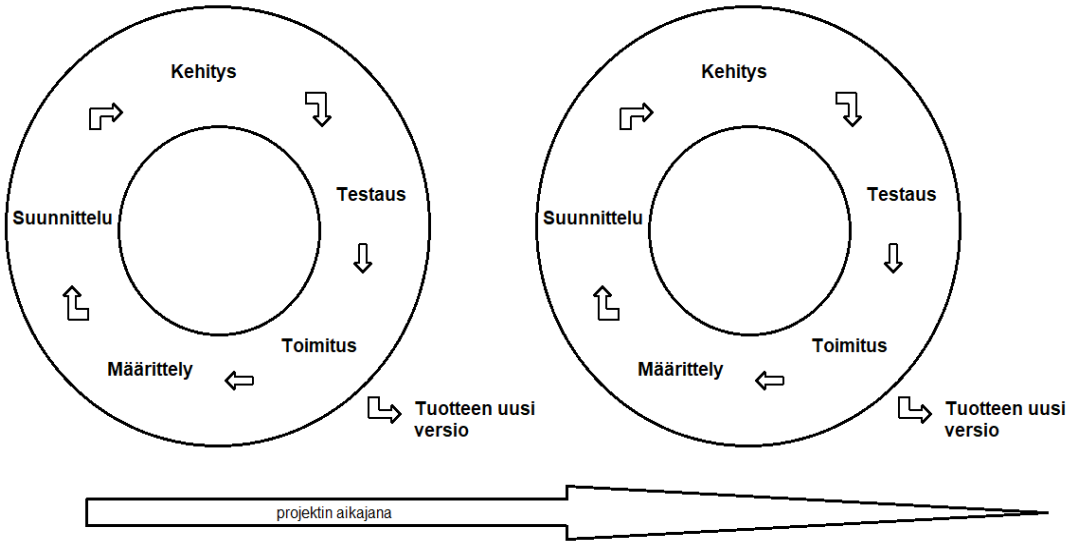
Ketterä ohjelmistokehitys on nykyaikaisen ohjelmistokehityksen suuntaus, jota käytetään yhä enemmän ohjelmistoteollisuudessa. Vuonna 2001 julkaistussa Ketterän ohjelmistokehityksen julistuksessa (Agile Manifesto) [4] ketterä kehitys määriteltiin kahdellatoista periaatteella. Ketterän kehityksen periaatteet on esitetty Liitteessä A. Määriteltyjen periaatteiden mukaan ketterästi toimiva kehitystiimi tyydyttää asiakkaan tarpeet toimittamalla ohjelmasta tarpeen täyttäviä versioita jo aikaisessa vaiheessa kehitystä ja säännöllisesti kehityksen aikana, suosien lyhyempää aikaväliä. Julistuksen mukaan projektin edistymisen ensisijainen mittari on toimitettavan ohjelmiston toimivuus.

Ketterässä ohjelmistokehityksessä siis toimitetaan asiakkaalle kehityksen alusta asti jatkuvasti uusia versioita ohjelmasta ja näin asiakas voi olla mukana vaikuttamassa kehityksen kulkuun ja mahdolliset epäkohdat tai monitulkintaisen määrittelyn aiheuttamat ongelmat voidaan huomata jo aikaisin kehityksessä. Perinteisessä vesiputousmallin mukaisessa projektissa (kuva 1) ohjelmiston alkuperäisen määrittelyn oikeellisuus on todella tärkeää, sillä määrittelyn ja suunnittelun jälkeen koko projekti toteutetaan näillä esitiedoilla ja pahimmassa tapauksessa mahdollinen virhe selviää asiakkaalle vasta aivan projektin lopussa. Nykyään suositaankin yhä enemmän ketteriä menetelmiä, joiden avulla muuttuviin määrittelyihin voidaan reagoida nopeasti ja tehokkaasti ja asiakkaan kanssa voidaan tehdä entistä lähempää yhteistyötä [5].



Kuva 1: Ohjelmistokehityksen vesiputousmalli

Ketterä kehitys suosii vesiputousmallin sijaan iteratiivista mallia (kuva 2), jossa projektin työmäärä jaetaan pienempiin osiin, jotka toteutetaan omina kokonaisuuksinaan. Iteraation aikana suoritetaan iteraatioon määrätyille tehtäville/ominaisuuksille määrittely, suunnittelu, kehitys, testaus ja toimitus kuten vesiputousmallin projektillekin, mutta iteratiivisessa kehityksessä toimitetaan asiakkaalle jo mahdollisimman aikaisessa vaiheessa kehitystä jonkinlainen toimiva pohja sovelluksesta. Iteraation lopussa voidaan saada asiakkaalta jo palautetta toteutuksesta ja tätä palautetta voidaan hyödyntää seuraavan iteraation suunnittelussa.



Kuva 2: Ketterän ohjelmistokehityksen malli

2.1.1 Scrum

Eräs laajalti käytetty [6] ketterän kehityksen työkalu on projektinhallinnan viitekehys Scrum. Scrum-viitekehyksen mukaisessa ohjelmistokehitysprosessissa projektihenkilöstöstä muodostetaan Scrum-tiimejä, joissa jokaisessa on henkilöstölle jaetut roolit ja projekti jaetaan ajallisesti pienempiin yhtä pitkiin kokonaisuuksiin (sprint), joiden välissä voidaan evaluoida projektin etenemistä ja suunnitella seuraavaa vaihetta.

Scrum-tiimi koostuu tuoteomistajasta, kehitystiimistä ja Scrum masterista. Tuoteomistajan velvollisuus on maksimoida tuotteen arvo kehitystiimin työn avulla. Tuoteomistaja hallitsee tuotteen kehitysjonoa, johon on määritelty mitä tehtäviä kehitystiimin tulee suorittaa iteraation valmistumiseksi. Kehitystiimi koostuu ammattilaisista, jotka toteuttavat tuotteen määritellyt ominaisuudet ja valmiin version tuotteesta iteraation päätteeksi. Kehitystiimi on itseohjautuva joukko, jolla on tarvittava osaaminen tuoteversion toteuttamiseksi. Scrum master on vastuussa Scrum menetelmien toteuttamisesta ja auttaa tiimiä toteuttamaan Scrum menetelmää. Scrum master toimii yhdessä tuoteomistajan kanssa toteuttaakseen Scrum-tapahtumia ja pitääkseen tuotteen kehitysjonon mahdollisimman tehokkaassa muodossa. [7]

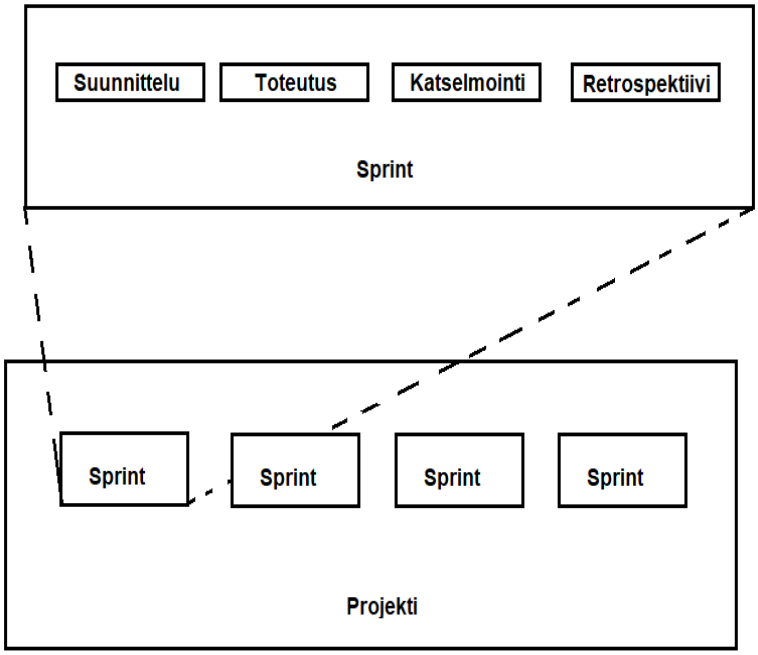
Tuotteen kehitysjono (product backlog) on järjestetty lista projektissa tarvittavista ominaisuuksista/suoritettavista tehtävistä. Tuotteen kehitysjono on projektin määrittelyn tärkein osa, mutta toisin kuin vesiputousmallissa, kehitysjono on dynaaminen ja sitä voidaan muokata responsiivisesti kehityksen aikana saadun palautteen tai ennalta näkemättömien

ongelmien perusteella. Kehitysjonossa on määritelty kaikki tulevat muutokset, ominaisuudet ja tehtävät, jotka tulee suorittaa tulevissa julkaisuissa. Tuoteomistaja on vastuussa tuotteen kehitysjonon ylläpidosta, järjestyksestä, saatavuudesta ja sen oikeellisuudesta. [7]

Scrum-tapahtumat on luotu tuomaan säännöllisyyttä ja minimoimaan muiden tapahtumien tarvetta projektin aikana. Kaikki Scrum tapahtumat ovat aikataulutettuja niin, että tapahtumilla on maksimikesto. [7]

Sprint on määritelty tapahtuma, joka toteuttaa yhden projekti-iteraation. Sprintin pituus on maksimissaan yksi kuukausi ja sprinttien pituus pidetään projektin aikana yhtenäisenä (Esimerkiksi 6kk pituisessa projektissa voi olla 6 kuukauden pituista sprinttiä tai 12 kahden viikon pituista sprinttiä). Sprint muodostuu suunnittelusta, päivittäisistä tapaamisista, kehitystyöstä, katselmoinnista ja retrospektiivistä. Sprintillä on aina tietty tavoite ja päämäärä ja sprintin tuloksena saadaan tuotteesta uusi tuoteversio. [7]

Sprintin sisältö päätetään sprintin alussa suunnitteluvaiheessa, joka on määrätty olevan maksimissaan 8-tuntinen tapahtuma. Suunnitteluvaiheessa koko Scrum-tiimi määrittää yhdessä mitä voidaan toimittaa alkavan sprintin aikana ja miten työ kannattaa jakaa, jotta tähän tavoitteeseen päästään. Päivittäinen Scrum-tapaaminen on 15 minuutin pituinen palaveri, joka pidetään sprintin jokaisena päivänä. Tässä tapaamisessa käydään läpi eteneminen ja tilanne kohti sprintin päämäärää. Sprintin katselmointi on maksimissaan neljätuntinen tapahtuma, joka pidetään sprintin lopussa. Katselmoinnissa käydään läpi uusi tuoteversio ja mahdollisesti päivitetään tuotteen kehitysjonoa vastaamaan nykyistä tilaa. Katselmoinnin ja seuraavan sprintin suunnitteluvaiheen välissä pidetään retrospektiivi, jonka tarkoituksena on olla mahdollisuus tarkastella menneen sprintin prosesseja ja keksiä mahdollisuuksia prosessien parantamiseen seuraavaa sprinttiä varten. Kuvassa 3 on esitetty sprintin sisältö. [7]



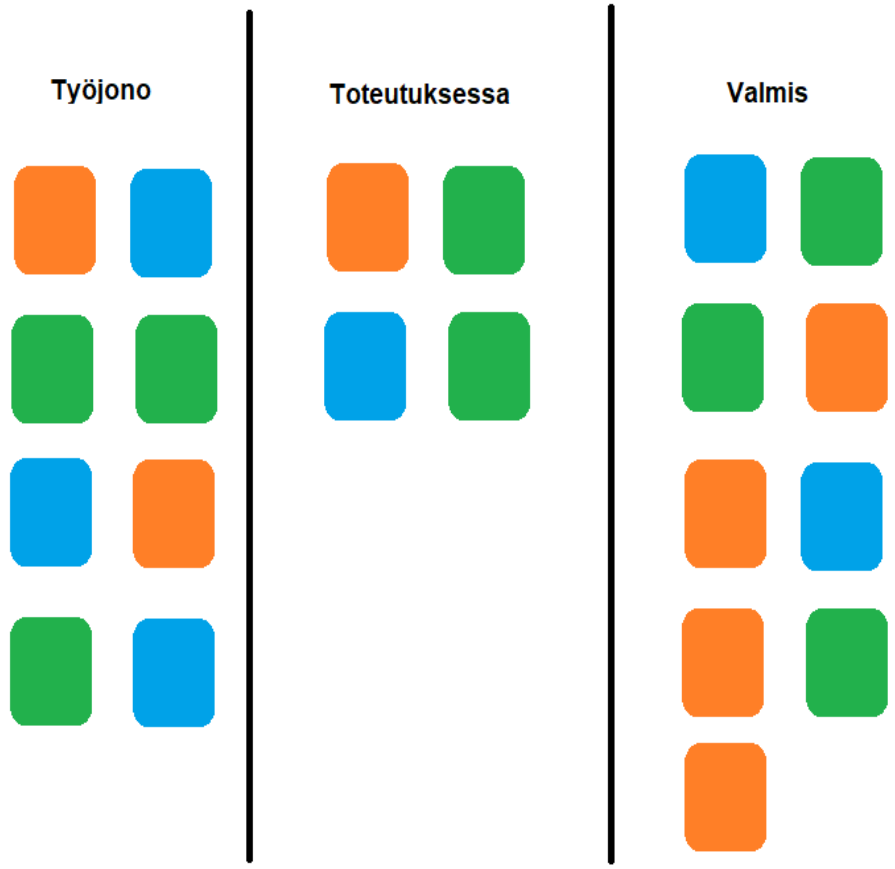
Kuva 3: Esitys Scrum-projektin rakenteesta

Pelkkä Scrum-menetelmän käyttö ei kuitenkaan itsessään takaa projektin onnistumista ja tavoitteisiin pääsyä. Scrum vaatii tiimiltä menetelmälle omistautuneisuutta, jotta sprintit pysyvät määritellyn mittaisina ja tavoitteissa pysytään. Scrum-tapahtumat on pidettävä säännöllisesti ja viitekehystä on käytettävä aktiivisesti. Esimerkiksi vaihtelevat työajat ohjelmistoalalla voivat haitata päivittäisten Scrum-tapaamisten toteutumista, jos kaikki tiimin jäsenet eivät pääse paikalle. [6] Tämän takia menetelmän toteutumisen arviointi retrospektiivissä on tärkeää.

2.1.2 Kanban

Kanban on Japanissa Toyotan autotuotantoa varten kehitetty ajoitusjärjestelmä, jonka avulla voitiin vähentää tekeillä olevaa työn määrää, monitoroida tuotantoprosesseja ja visualisoida tuotannon ajoituksia. Kanban järjestelmässä käytetään Kanban-kortteja tiettyjen tuotannon prosessien ajoittamiseksi. Prosessin eri osiin liittyvien korttien kerääntymisellä saatiin visualisoitua tuotantoa reaaliajassa. [8] Kanbanin tehtävänä on osoittaa tarkasti, mitkä tehtävät on tehtävä ja milloin ne on tehtävä. Tämän Kanban ratkaisee priorisoimalla tehtäviä ja määrittämällä työnkulkua. Kanbanin periaatteiden mukaan keskeneräisiä työtehtäviä tulisi olla jatkuvasti mahdollisimman vähän, työn etenemisen pitäisi olla näkyvää ja työjonojen pitäisi olla ennalta suunniteltuja. [9] Työjonojen, etenemisen

ja kesken olevien töiden visualisoimiseksi käytetään Kanban-taulua, jolla työtehtävät on esitetty kortteina. Yksinkertainen ohjelmistokehityksessä käytettävä Kanban-taulu on esitetty kuvassa 4.



Kuva 4: Kanban-taulu

Kanban-taululla iteraation työtehtäville on tehty Kanban-kortit, jotka on asetettu taululle kategorioittain tehtävän tilan mukaan. Kun kehittäjä aloittaa tehtävän toteutuksen, hän siirtää tehtävän kortin ”Työjono”-osiosta ”Toteutuksessa”-osioon ja tehtävän valmistuessa kortti siirretään ”Valmis”-osioon. Kanban-taulu on projektin työtehtävien hallinnan lisäksi visuaalinen esitys projektin tehtävien tilasta. Taulu voidaan toteuttaa esimerkiksi tarralapuilla seinälle tai virtuaalisesti jonkinlaisen jaetun verkkotyökalun avulla. Ajan tasalla pidetystä Kanban-taulusta kehitystiimin jäsenet näkevät suoraan valmiina, toteutuksessa ja tekemättä olevien työtehtävien määrän ja taulu tukee työn edistymisen visualisointia työntekijöille. [8]

Kanbania voidaankin käyttää esimerkiksi Scrum-viitekehyksen tukena asettamalla sprintin alussa sprinttiin valittujen tehtävien kortit taulun ”Työjono”-osioon ja onnistuneen sprintin lopussa kaikkien korttien pitäisi olla ”Valmis”-osiossa. Niinpä Kanban-taulu voi

olla hyvä tuki esimerkiksi päivittäisten Scrum-tapaamisten tueksi, jolloin taululta nähdään suoraan mitä kukin on tekemässä ja paljonko työtä on tehty ja paljonko sitä on jäljellä.

Kuvan 4 esimerkin kolmen osion lisäksi taululle voidaan tarvittaessa lisätä omat osiot esimerkiksi testaukselle tai tehtävän toiminnallisuuden verifioinnille. Näiden osioiden avulla tehtävän toteutus ja testaus voidaan eristää toisistaan. Eri kategoriat tulee toki valita projektitiimille ja projektille sopiviksi.

Kanban-menetelmän mukaisia työjonoja ja työtehtävien tiloja visualisoivia tauluja voi luoda ja ylläpitää myös virtuaalisesti. Esimerkiksi Trello [10] tarjoaa tehtävienhallintaan alustan, jolla voidaan toteuttaa Kanban-tyylisiä tauluja. Trello tarjoaa myös sovellusintegraatiot esimerkiksi Github ja Jira -alustoihin, jolloin eri työkalujen käyttö samassa projektissa onnistuu helpommin.

2.2 Ohjelmistokehityksen työkalut

Ohjelmistokehityksen ja etenkin ketterän kehityksen tukena käytetään paljon iteratiivisia menetelmiä tukevia pilvipohjaisia projektinhallinnan työkaluja, joiden avulla projektin kehityksen etenemistä voidaan suunnitella ja dokumentoida. Näiden työkalujen tarkoituksena on koostaa tuotteen toiminnallisuuden, ominaisuuksien ja tehtävien määrittelyjä ja helpottaa näiden käsittelyä. Ketterässä kehityksessä esimerkiksi asiakkaan edustajat voivat myös lisätä ja muokata määrittelyjä projektin aikana erilaisilla projektinhallintaan ja määrittelyihin liittyvillä alustalloilla ja näin asiakkaan vaatimukset saadaan suoraan muun spesifikaation kanssa samalle alustalle, ilman että kehitystiimi joutuu näitä parsimaan esimerkiksi sähköpostiviesteistä tai asiakastapaamisten muistiinpanoista.

Tehtävienhallintaohjelmistot auttavat kehitystiimiä määrittelyiden, työtehtävien jakamisen ja tehtävien tilojen hallinnan manageroinnissa. Jira [11] on tehtävienhallinta-alusta, jonka avulla voidaan hallita suurta määrää projektin valmistumisen vaatimia osatehtäviä. Tehtävien etenemisen tilaa voidaan seurata, tehtävien vaikutuksia ja riippuvuuksia voidaan tarkastella ja tehtävien sisällöstä voidaan käydä keskustelua alustalla. Tehtäviä voidaan luoda alustalle lisää projektin edetessä ja olemassa olevia tehtäviä voidaan muokata ja laajentaa tarvittaessa.

Versionhallintajärjestelmät ovat tärkeä osa modernia ohjelmistokehitysprosessia. Kehitysprosessin aikana havaitut bugit, määrittelyiden muutokset ja tarvittavat uudet ominaisuudet johtavat lähdekoodin muutoksiin. Muutoksia tehtäessä usean henkilön projektissa, on tärkeää, että kehitystiimin jäsenet tietävät, miksi jokin osa ohjelmaa on muuttunut ja minkä takia. Usein voidaan myös törmätä tilanteisiin, joissa tehtyjä muutoksia halutaan

syystä tai toisesta peruuttaa tai ottaa käyttöön aiemmin poistettuja ominaisuuksia tai niihin liittyvää koodia. Näissä tilanteissa on tärkeää, että ohjelmakoodista on olemassa versiohistoria, josta nähdään ohjelmakoodin vanhemmat versiot ja muutokset. Versionhallintajärjestelmää käytettäessä aina kun lähdekoodiin on tehty jonkinlainen pysyvät muutos (esimerkiksi uusi ominaisuus tai korjaus aiempaan), tekee kehittäjä versionhallintajärjestelmään merkinnän muutoksesta ja lisää kommentin, joka selittää mitä muutoksessa on tehty ja miksi se on tehty. Versionhallintajärjestelmä tallentaa projektin lähdekoodien sen hetkisen tilan ja versiohistoriaan jää kommentin kera tieto siitä, missä tilassa lähdekoodi on ollut tämän muutoksen hetkellä. Muutoksen jälkeen projektin tila voidaan päivittää keskitettyyn versionhallintajärjestelmän alustaan, josta kehitystiimin muut jäsenet voivat ladata tuoreimman version lähdekoodista. Versionhallinta-alustan avulla voidaan päästä käsiksi kaikkiin projektin tiedostojen aiemmin versioituihin versioihin ja lisäksi näiden kommenttien avulla versiohistoria dokumentoituu alustalle. [12]

Git on Linus Torvaldsin vuonna 2005 kehittämä hajautettu versionhallintajärjestelmä. Hajautetun toimintatapansa vuoksi jokainen Git repositorio sisältää projektin täyden versiohistorian, eikä käyttäjän tarvitse olla yhdessä versionhallintapalvelimeen saadakseen käsiinsä ohjelman vanhempia versioita. Uusia versioita ja muutoksia voidaan myös tehdä paikallisesti, vaikkei yhteyttä projektin keskitettyyn repositorioon ole ja muutokset voidaan myöhemmin ”puskea” käytetylle Git-palvelimelle. Git on yhä suositumpi valinta projektin versionhallintajärjestelmäksi hajautetun arkkitehtuurinsa ja suurien verkkopalvelualustojen Git-tuen vuoksi. [13]

Github [14] on Gitillä toimiva verkkopohjainen versionhallinta-alusta lähdekoodille, jonka avulla voidaan taltioida ohjelmiston versiohistoria ja lähdekoodi Githubin palvelimille. Ohjelmakoodin versionhallinnan lisäksi Github tarjoaa projektille wiki-sivun, johon voidaan kirjata ohjelmiston käyttöön liittyviä ohjeita, ohjelmiston jatkokehitykselle merkityksellisiä asioita tai ohjelman kehityksen vaiheita. Lisäksi alusta tarjoaa mahdollisuudet projektin bugien ilmoittamiseksi ja uusien toiminnallisuuksien määrittelemiseksi, ja muita ominaisuuksia projektinhallintaan.

Jatkuvan integroinnin tarkoituksena on integroida toteutettua työtä ja uutta ohjelmakoodia ohjelmistontuotantoprosessissa jatkuvasti, jotta mahdolliset integraatio-ongelmat pysyisivät mahdollisimman pieninä. Avainasemassa jatkuvassa integraatiossa on keskitetty versionhallintarepositorio, johon kehittäjät lisäävät tekemänsä muutokset mahdollisimman usein. Jatkuvan integroinnin palvelimen tehtävänä on verifioida ohjelmakoodin ja ohjelman toimivuutta kehityksen aikana. Jatkuvan integroinnin palvelin voi esimerkiksi hakea automaattisesti päivitetyt tiedostot versionhallinta-alustalta, kun uusia muutoksia tehdään. Tämän jälkeen palvelin voi kääntää ohjelman automaattisesti ja ajaa tälle määritellyt testit. Käännösvirheen tai testien epäonnistumisen seurauksena palvelin voi ilmoittaa kehitystiimille esimerkiksi sähköpostilla tai muulla välineellä tapahtuneesta vir-

heestä ja kehitystiimi saa välittömästi tiedon ongelmasta. Jatkuvan integraation menetelmien on tarkoitus tarjota jatkuva ja puolueeton tieto kehityksen tilasta ja ohjelmakoodin toimivuudesta. [15]

3. VISUALISOINTI

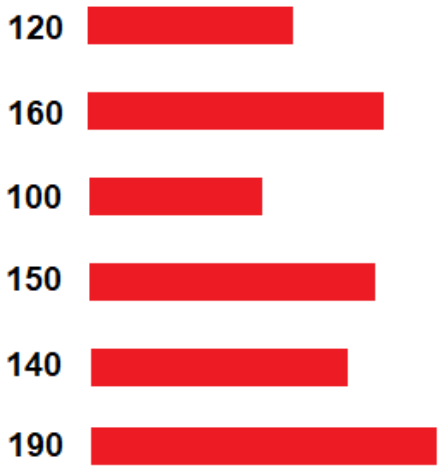
Yksi kuva kertoo enemmän kuin tuhat sanaa. Tämä johtuu siitä, miten ihmisen näkö toimii. Ihmisen näköaisti on erittäin herkkä näkemään eroja koossa, muodoissa, väreissä ja asennoissa. Niinpä esimerkiksi täysin abstrakti numeromuotoinen informaatiotulva voidaan esittää kokoelmana erilaisia muotoja ja värejä, jotka esittävät näitä vastaavaa informaatiota esimerkiksi kokonsa tai muotonsa avulla muiden objektien suhteen, jolloin datasta voi saada kattavan kokonaiskuvan jopa nopealla vilkaisulla. Lisäksi kuvamuotoinen informaatio helpottaa asioiden muistamista ja vauhdittaa ihmisen kognitiivista toimintaa. [16]

3.1 Visualisoinnin menetelmät

Visualisointi pyrkii esittämään tietoa uudessa muodossa, niin että se on helpommin sisäistettävissä. Visualisoidessa raaka data voidaan muuttaa kuviksi, graafeiksi ja konkreettisiksi muodoiksi esimerkiksi pelkkien numeraalisten arvojen sijaan, jolloin tieto on lukijalle heti ymmärrettävämmässä muodossa. Dataa visualisoidessa kolme tärkeää päämäärää tuotettavalle visualisaatiolle ovat sen ilmaisukyky, tehokkuus ja tarkoituksenmukaisuus. [17]

Maailmassa on yhä enemmän informaatiota ja dataa, joita meidän tulee sisäistää päivittäin. Kaiken tämän datan käsittelemiseen käytetään tiedon sisäistämisen helpottamiseksi graafisia esityksiä eli visualisointeja. [18] Visualisointia käytetään nykymaailman datatulvan apuna jatkuvasti, esimerkiksi päivän lehdessä pörssikurssien graafit kuvaavat osakkeen arvon muutoksia ajan suhteen pelkän arvojen listausten perusteella. Pienemmässä tilassa arvo voidaan ilmoittaa pelkkänä prosenttimuutoksena, mutta tällöinkin negatiiviset luvut on esitetty punaisella ja positiiviset vihreällä. Kurssimuutokset on myös järjestetty sen mukaan, mikä kurssi on muuttunut eniten ylöspäin ja mikä eniten alaspäin. Näin yksinkertaisesta listasta sana- ja lukupareja on muodostettu visualisaatio värien ja järjestämisen avulla. Sääennusteessa kartalle on asetettu suurien kaupunkien kohdalle ikoneita, jotka ilmoittavat päivän säätilan sateisesta aurinkoiseen ja päivän ylimmän lämpötilan.

Kuvassa 5 on esitetty sarja numeroarvoja ja visuaalinen esitys arvoista pylväinä, joiden pituudet on suhteutettu niiden esittämiin arvoihin.



Kuva 5: Numeeristen arvojen esitys pylväinä

Jos vasemmalla esitetyistä numeroarvoista tulisi löytää pienin ja suurin, eikä arvoja vastaavia pylväitä olisi esitettyinä, tyypillinen ratkaisu olisi lukea kaikki arvot ja pitää mielessä suurin ja pienin vastaan tullut arvo näitä keskenään vertaillen. Kun arvojen lisäksi on tarjottu visuaalinen esitys pylväiden pituuksien avulla, nähdään ensisilmäyksellä millä riveillä on pienin ja suurin arvo.

Hyvän visualisaation tekemiseksi on tärkeää ottaa huomioon kenelle visualisaatio tehdään ja mitä tarkoitusta varten se tehdään. Abstraktia dataa visualisoidessa on aina tärkeää ottaa huomioon mitä tietoa visualisaation on tarkoitus antaa sen katselijalle. [18] Jos esimerkiksi kuvan 5 numeroarvoista haluttaisiin löytää 20:llä jaolliset luvut, ei pylväiden avulla tuotettu esitetys olisi kovinkaan hyvä tai arvokas visualisaatio. Tähän käyttötapaukseen pylväät voisi esimerkiksi värittää eri värillä arvojen 120, 160, 100 ja 140 kohdalla ja jos arvojen suuruudella ei ole merkitystä, palkkeja ei tarvitsisi esittää lainkaan, vaan arvon vierellä voisi olla jokin kompaktimpi muoto, jonka värillä tuotaisiin esille huomionarvoiset arvot.

Visualisoitava data tulee kuitenkin ensin koostaa ja muotoilla oikeanlaiseen, muotoon, jotta visualisaatio saadaan muodostettua. Mikä tahansa datasarja voidaan piirtää yksinkertaisin säännöin kuvaksi, mutta jotta visualisaatiosta saataisiin ilmaisukykyinen, tehokas ja tarkoituksenmukainen, tulee käytettävän informaation datamalli suunnitella tarkasti. Tarkoituksenmukainen datamalli helpottaa datan visualisointia ja tukee valittua esitystapaa. [18]

3.2 Visualisointi ohjelmistokehityksessä

Visualisoinnilla on tärkeä rooli ohjelmistoanalytiikassa, jonka päämääränä on ratkaisut tehokkaampiin ohjelmistoprojektien projektinhallintaratkaisuihin. Mattilan et al. suorittaman kirjallisuuskatsauksen mukaan kuitenkin suuri osa lähivuosina julkaistuista artikkeleista ohjelmistojen visualisoinnista keskittyy ohjelmistojen rakenteen ja käyttäytymisen tutkimiseen, eikä esimerkiksi ohjelmistokehityksen prosessien visualisoinnista ole juuri tehty tutkimusta. [19]

Stephan Diehlin määritelmä ohjelmistojen visualisoinnista kirjassaan on: ”*Ohjelmistojen ja niiden kehitykseen liittyvien artifaktien visualisointi”.* Artifakteilla viitataan tässä paitsi ohjelmistoihin ja niiden ohjelmakoodiin, myös ohjelmistokehitykseen liittyviin määrittelyihin, dokumentaatioon ja lähdekoodin muutoksiin. Diehl jakaa ohjelmistojen visualisoinnin myös kolmeen eri kategoriaan. Rakenteellinen visualisointi visualisoi ohjelmiston staattisia elementtejä, joita voidaan analysoida ilman ohjelman suoritusta. Esimerkiksi ohjelmiston lähdekoodi, ohjelman tietomallit ja ohjelman rakenteeseen liittyvät asiat kuuluvat rakenteelliseen visualisointiin. Toinen kategoria on ohjelman käyttäytymisen visualisointi. Käyttäytymisen visualisointiin kuuluu ohjelman suorituksen visualisointi erilaisilla syötteillä. Kolmas Diehlin määrittelemä kategoria on ohjelmiston evoluution visualisointi. Tämä viittaa ohjelmistojen kehitysprosessien ja erityisesti ohjelmakoodin muutosten visualisointiin. Näistä kategorioista erityisesti ohjelmistojen evoluution ja ohjelmistoprosessien visualisointi on merkittävää tämän työn kannalta. [20]

Paredes et al. mukaan olemassa olevissa ohjelmistokehityksen aikana käytettävissä prosessien visualisaatioissa hyöty projektille tuotetaan useimmiten tuomalla selkeyttä sen etenemiseen ja tarjoamalla välitöntä palautetta. Projektin aikana käytettäviä visualisaatioita voivat olla esimerkiksi jatkuvan integroinnin alustan avulla tuotetut kuvaukset tuotteen toiminnasta, joiden avulla nähdään, jos esimerkiksi jonkin ohjelman suoritus on keskeytynyt virheeseen automaattisten testien ajossa. Tämän tyyppisiin visuaalisiin avusteisiin käytetään usein työtilassa jatkuvasti näkyvillä olevaa näyttöä, tai muuta dynaamista esinettä, joka ilmaisee projektin aikana tärkeitä asioita, kuten projektin työmäärän toteutumista tai testiympäristöjen tuloksia. [21]

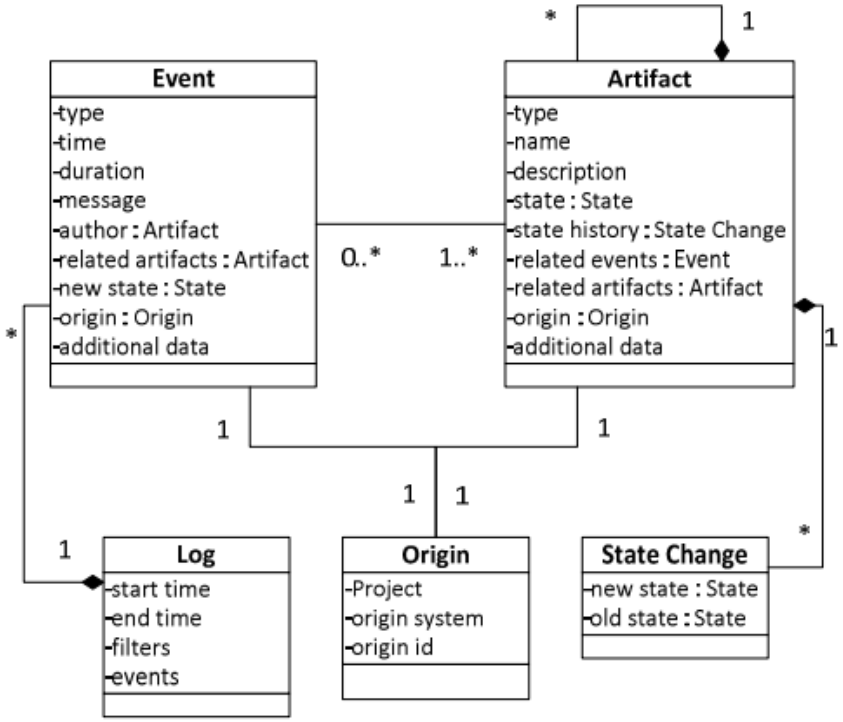
Projektin etenemisen mittareita käytetään visualisoinneissa, jotta saadaan konkreettinen kuva työn etenemisestä ja työjonon muutoksien vaikutuksesta työmäärään. [20] Burn up- ja burn down -kaaviot näyttävät kuinka paljon työtä on suoritettu tietyn aikavälin aikana (burn up) ja kuinka paljon työtä on jäljellä tietyn tavoitepisteen saavuttamiseksi (burn down). Näiden työmääriä kuvaavien kaavioiden käyttö on yleistä ja esimerkiksi Jira-tehtävienhallintaohjelmisto käyttää näitä projektin etenemisen visualisoimiseksi.

Brandt et al. on tutkinut ja kehittänyt Essence-viitekehyksen menetelmien pohjalta visualisointialustan, joka toteuttaa yleisnäkymän projektin tilaan. Sovelluksessa ohjelmistokehitysprosessi on jaettu erilaisiin vaiheisiin ja kategorioihin, joille on määritelty tietyissä

vaiheissa suoritettavia projektin etenemiseen liittyviä tarkistuslistoja. Alusta kuitenkin vaatii käyttäjäsyötettä projektin etenemisen seuraamiseksi, joten periaatteessa se toteuttaa vain uuden projektinhallinta/seuranta-alustan, joka tuottaa yhä enemmän hajautettua dataa projektista. [22]

Ohjelmistotuotantoprosessin retrospektiivista visualisointia varten täytyy projektiin liittyvää dataa saada jostain visualisoitavaan muotoon. Projektin etenemisen tietojen koostaminen ja näiden tietojen muotoilu käsin olisi erittäin työlästä ja tulisi näin hyvin kalliiksi. Tämän takia projektiin liittyvä data kannattaakin kerätä käytettävistä työkaluista automatisoidusti. Useat modernit alustat tarjoavatkin sisältämiensä tietojen käsittelyä varten ohjelmointirajapintoja, joista tietoja voidaan lukea ohjelmallisesti. Näiden ohjelmointirajapintojen hyväksikäyttö ja datan kerääminen on ensimmäinen askel projektidatan visualisointia kohti. Eri alustat voivat kuitenkin sisältää keskenään hyvinkin erilaista dataa ja tiedon jäsennyksessä ja muodossa on todennäköisesti yhtä monta tyyliä, kuin on palvelujakin. Tämän takia kerätty data tulee koostaa yhtenäiseen muotoon, jolloin useasta alustasta koostettua dataa voidaan hyödyntää yhtenäisessä analyysissa.

Mattila et al. on ehdottanut ohjelmistoprojektidatalle yhteistä tietomuotoa, jossa data jaetaan tapahtumiin (Event) ja artifakteihin (Artifact). Tässä mallissa tapahtuma on jokin ohjelmistokehitykseen liittyvä tapahtuma, jolla on tapahtuma-aika, kesto ja tekijä. Artifakti taas on mikä tahansa ohjelmistokehitykseen liittyvä asia, joka voi olla mielenkiintoinen projektin analysoinnin kannalta. [3] Mattilan et al. ehdottama ohjelmistoprojektidatan datamalli on esitetty kuvassa 6.



Kuva 6: Mattilan et al. ehdottama datamalli ohjelmistoprojektiin liittyvälle datalle [3]

Cosentino et al. on myös tutkinut ohjelmistoprojektissa käytettävien työkalujen datan koostamista ja keskittämistä yhteisen tietokannan alle. Cosentinon et al. Gitana-alustan avulla voidaan koostaa projektinhallintadataa useasta eri lähteestä samaan relaatiotietokantaan. Työkaluun on toteutettu datankerääjäohjelmia useille eri projektinhallintaan liittyville alustoille, muun muassa Git, Bugzilla, Github ja Slack. Gitana-alusta toteuttaa kerätylle datalle yhteisen tietokantaskeeman ja tarjoaa käyttäjälle mahdollisuuden tutkia ja hakea dataa SQL-hakujen avulla. [23]

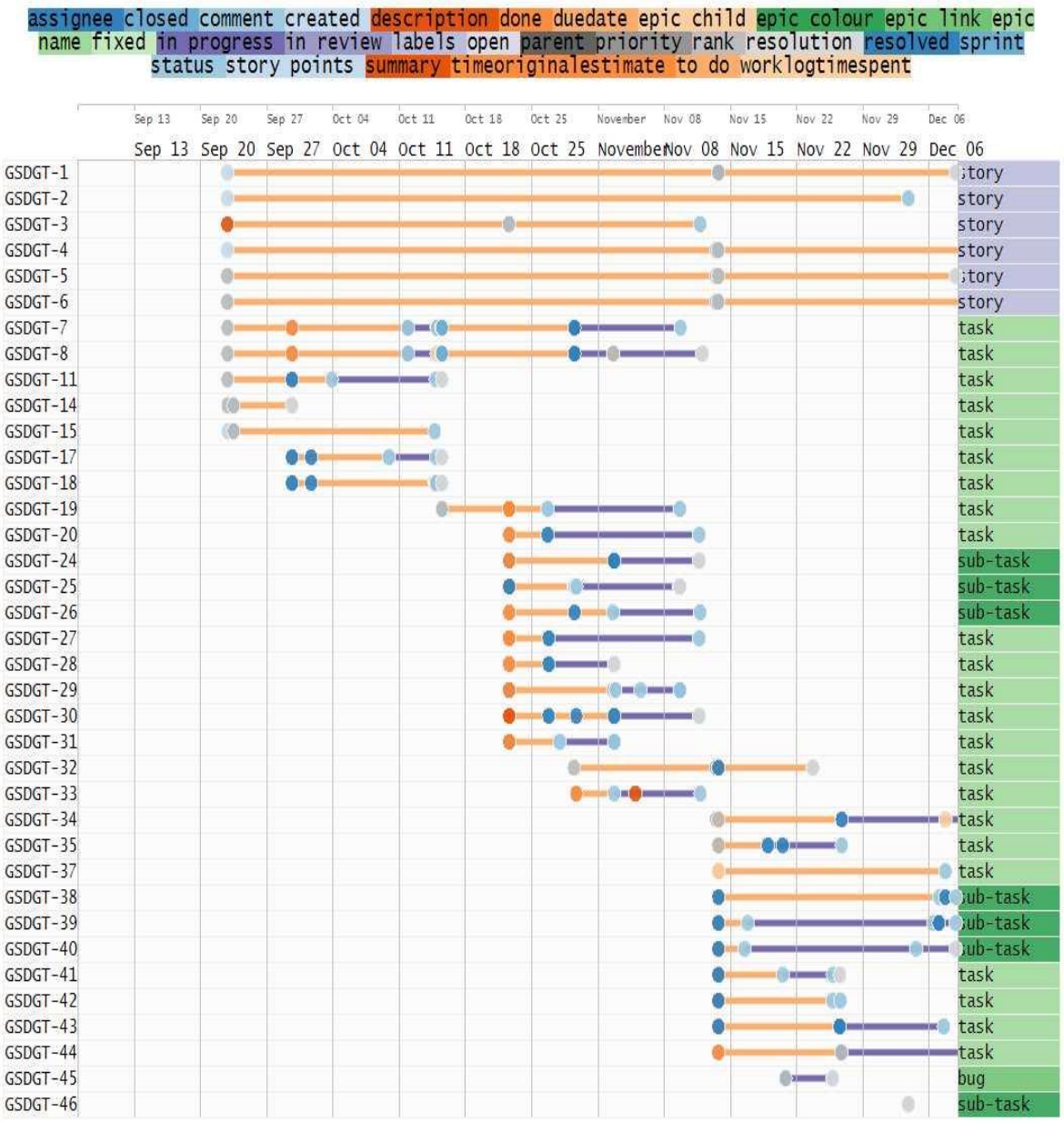
4. TAPAUSTUTKIMUKSEN TAUSTAT JA MÄÄRITTELY

Tapaustutkimuksena työn ohella suoritetaan olemassa olevaan projektinhallintadatan visualisointiin kehitettyyn työkaluun laajennus, jonka avulla työkalulla voidaan visualisoida dataa yhdestä uudesta lähteestä. Laajennus tukee paitsi työkalun lähdedatan mahdollisuuksia, myös yliopistolla tehtävää opetustyötä, sillä halutun alustaintegraation avulla voidaan työkalua käyttää kurssidatan visualisoinnissa ja arvioinnissa.

Tampereen Teknillisen Yliopiston opintojaksolla Software Engineering Methodologies opetetaan ohjelmistoprojektin ketterää kehitystä ja Scrum-viitekehyksen käyttöä kehityksen apuna. Scrumin vaatimaan tuotteen kehitysjonon ylläpitoon ja muuhun projektinhallintaan käytetään opintojaksolla Agilefant-nimistä projektinhallinta-alustaa. Osana opintojakson arvostelua on Scrum-menetelmän käyttö. Arvostelua voitaisiin helpottaa Agilefant-projektien manuaalisen selaamisen ja tarkastelun sijaan lukemalla projektin metatiedot automaattisesti hyödyntämällä Agilefant-alustan tarjoamaa ohjelmointirajapintaa ja jäsentämällä tästä datasta visualisaatioita, jotka tukisivat ennen kaikkea kokonaisuutena projektin etenemisen tarkastelua. Yliopistolla on aiemmin kehitetty visualisointiohjelmaa, joka tuottaa halutun tyyppisiä visualisaatioita projektinhallintadatasta, mutta ohjelmassa ei ole datankerääjäohjelmaa, jolla saataisiin luettua dataa Agilefanti-alustan projektidataa.

4.1 N4S-visu

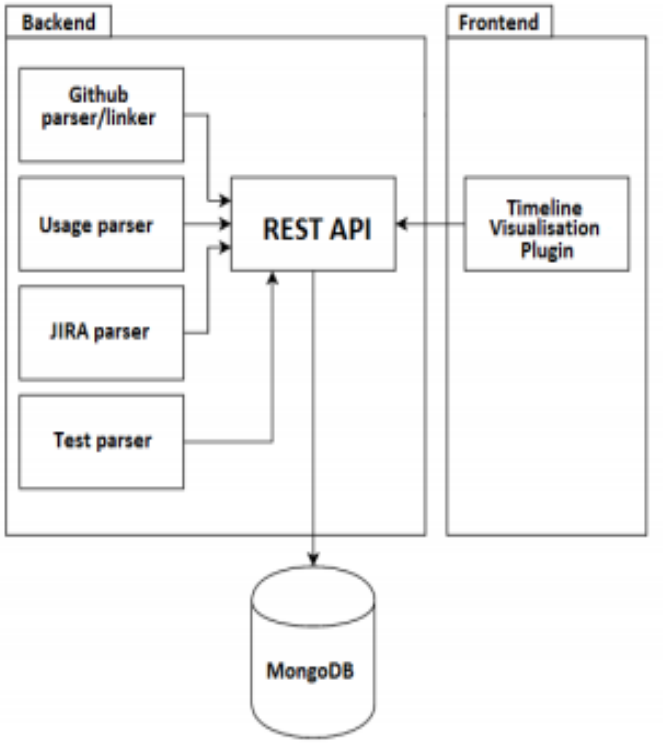
Need4speed vis tool (Tästä eteenpäin tässä dokumentissa n4s-visu) on Tampereen Teknillisellä Yliopistolla kehitetty projektinhallintadatan visualisointialusta, jonka avulla voidaan visualisoida esimerkiksi Gitin versionhallintadataa tai Jiran tehtävienhallintadataa. Ohjelmaan voidaan hakea datankerääjätyökalulla versionhallintadataa ohjelmallisesti suoraan tuettujen alusten dataa niiden tarjoamista ohjelmointirajapinnoista. Datan järjestelmään tallentamisen jälkeen sitä voidaan tarkastella verkkoselaimessa ja sitä voidaan suodattaa eri tavoin visualisointia varten. Kuvassa 7 on kuvattu esimerkki työkalulla luodusta visualisaatiosta, jossa on esitetty projektin tehtävien etenemistä ajan suhteen.



Kuva 7: n4s-visun generoima visualisaatio

N4s-visu on toteutettu Node.js Javascript-ympäristön avulla ja sen tietokantana toimii MongoDB-tietokanta. Palvelinohjelma pitää yllä tietokantaa ja tarjoaa REST-ohjelmointirajapinnan, jonka avulla dataa voidaan käsitellä. Visualisoitavaa dataa voidaan lisätä tietokantaan erilaisilla datankerääjäohjelmilla, jotka hakevat datan halutusta lähteestä (esim. Github), muotoilevat sen haluttuun muotoon ja lähettävät sen palvelimelle http post -kutsulla. Itse visualisoinnin hoitaa d3.js-kirjaston avulla toteutettu selainohjelma, joka hakee valitun datan palvelinohjelmalta ja piirtää sen käyttäjälle nähtäväksi.

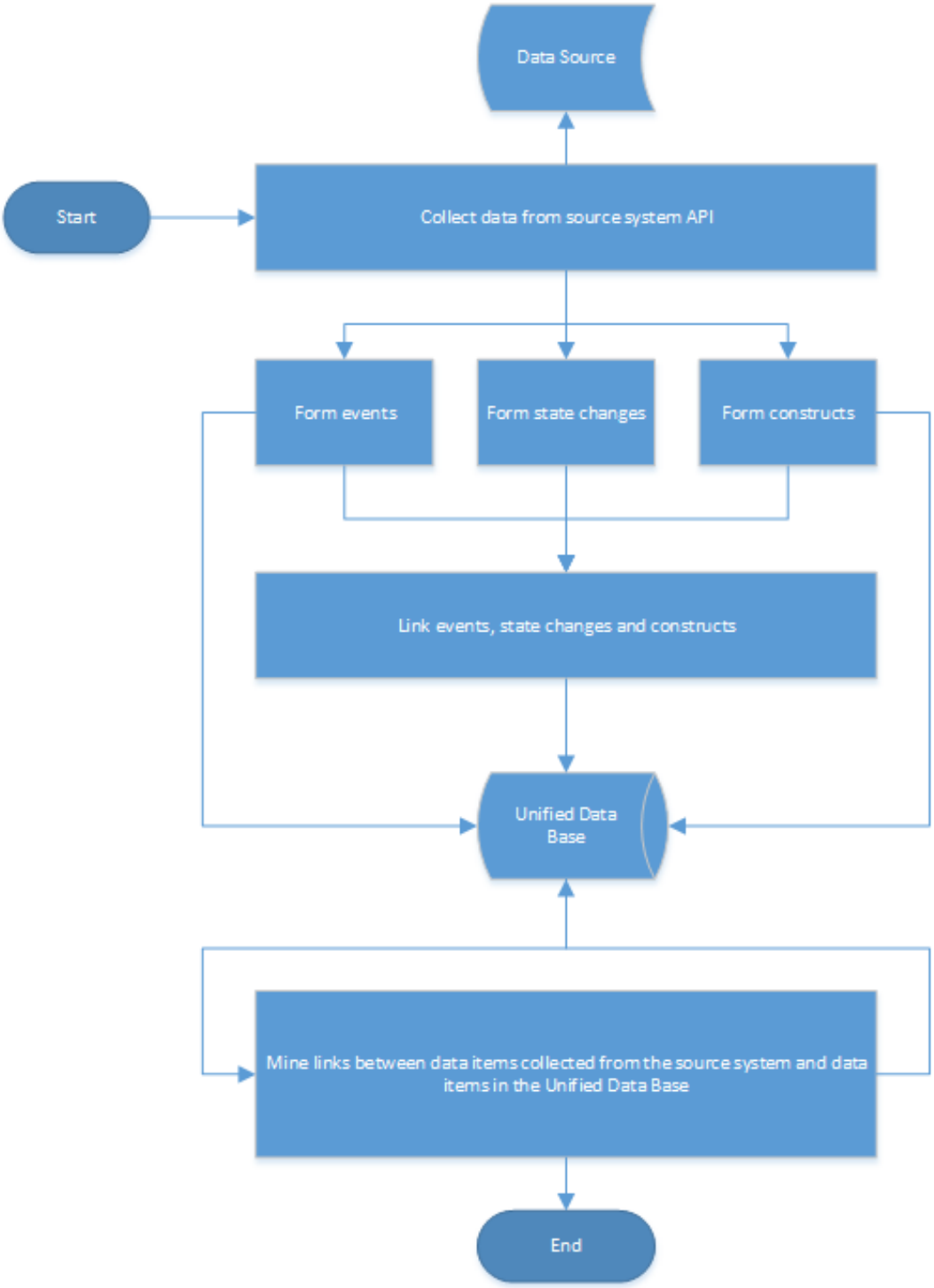
Kuvassa 8 on esitetty Mattilan et al. ehdottama [3] ja visualisointityökalun toteuttama arkkitehtuurimalli. Palvelinpuolella toimivat datankerääjäohjelmat lisäävät REST-rajapinnan avulla dataa MongoDB-tietokantaan ja asiakasohjelma hakee tätä dataa tietokannasta REST-rajapinnan kautta piirtääkseen visualisaatiot datasta.



Kuva 8: N4s-visun toteuttama arkkitehtuurimalli [3]

4.2 Visualisointityökalun datankeruumalli

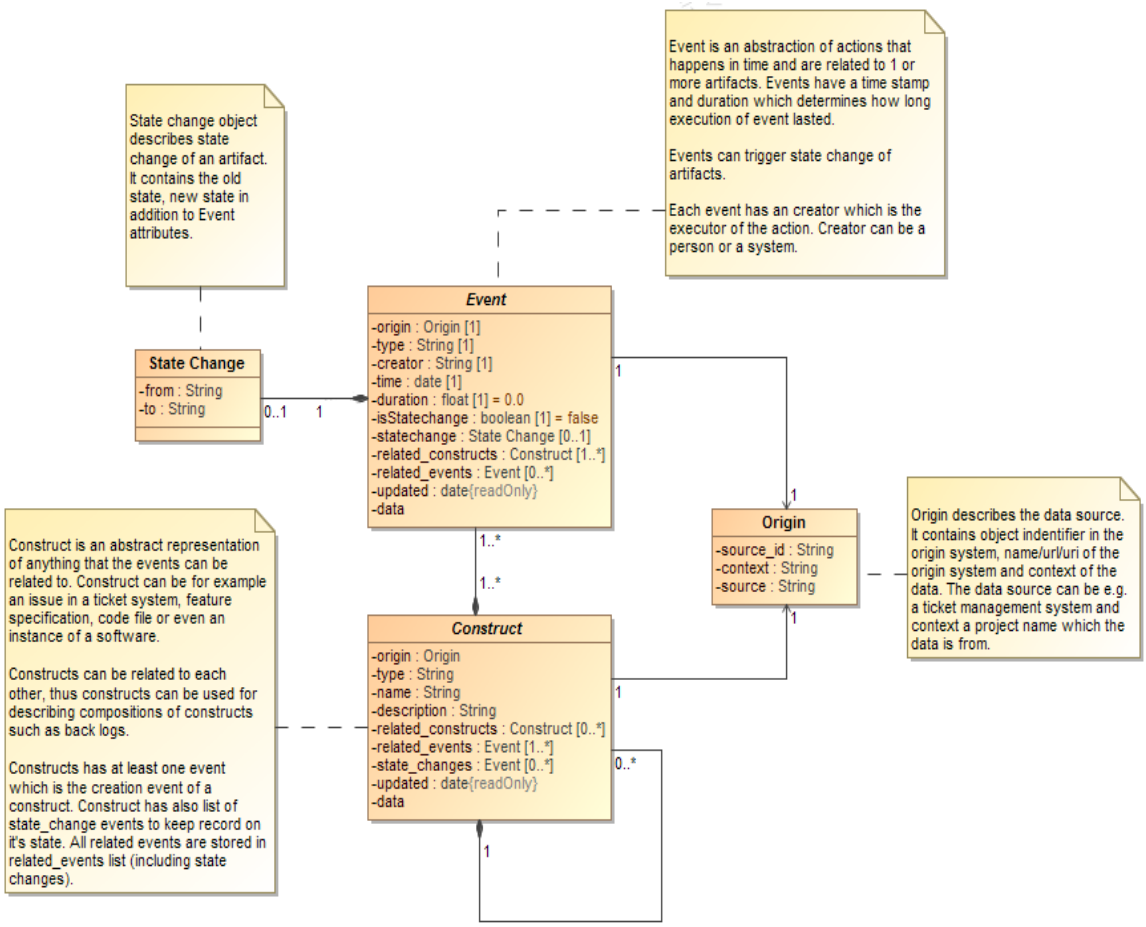
Kuvassa 9 on esitetty n4s-visun datankeruumalli ja datankerääjämoduulin toteutettavat vaiheet datan lisäämiseksi työkalun tietokantaan. Datankeruumallin mukainen ohjelma hakee halutusta lähteestä raa'an datan ja tämän jälkeen muotoilee siitä ohjelman tietokantaskeeman mukaisia objekteja ja tämän jälkeen tallentaa ne tietokantaan. Tallentamisen jälkeen olisi myös mahdollisuus linkittää dataa esimerkiksi jostakin toisesta tietolähteestä haettuun dataan, jolloin esimerkiksi JIRA-tiketit voitaisiin yhdistää Git commiteihin.



Kuva 9: n4s-visun datankeruumalli [24]

N4s-visun käyttämä datamalli perustuu Mattilan et al. ehdottamaan ohjelmistoprojektin yhteiseen tietomuotoon. [2] Ohjelman datamallissa data on jaettu tapahtumiin (event), tilanmuutoksiin (state change) ja konstruktioihin (construct). Konstruktio on entiteetti, johon tapahtumat liittyvät, eli esimerkiksi toteutettava tehtävä tai versionhallintaan mer-

kitty ongelma (esim. Git issue). Tapahtuma on tietotyyppi, jolla on tapahtuma-aika. Tapahtuma voi olla toteutettavaan tehtävään liittyvä versionhallinnan muutos (Git commit) tai tehtävän tilanmuutos (keskeneräisestä valmiiksi jne.). Tapahtuma liittyy aina konstruktioon ja tapahtuma voi olla myös tyypiltään tilanmuutos. Kuvassa 10 on esitetty työkalun datamalli.



Kuva 10: n4s-visun datamallin luokkakaavio [24]

Tapahtumalla ja konstruktiolla on myös alkuperä (Origin) -kenttä, johon on määritetty datan alkuperä ja sen konteksti. Esimerkiksi Git issuen alkuperä voi olla Github ja sen konteksti tietty Git-repositorio. Alkuperästä selviää, mitä dataa malli sisältää, ja kontekstin avulla voidaan eritellä usean eri kokonaisuuden datat toisistaan. Jos tietokannassa on esimerkiksi usean projektin tiedot Gitistä ladattuina, voidaan dataa suodattaa Git-repositorion nimellä, jolloin saadaan jäämään jäljelle vain yhden projektin data.

4.3 Agilefant

Agilefant on projektinhallintatyökalu, jonka avulla voidaan suunnitella, ylläpitää ja seurata projektin tilaa ja etenemistä. Työkalun avulla projekti voidaan jakaa ennalta määrätyn pituisiin iteraatiojaksoihin (esim. Scrum sprint) ja näihin iteraatioihin koostetaan iteraation aikana tehtävät käyttäjätarinat ja tehtävät.

Käytännössä yliopiston opintojaksolla käytettäessä Agilefantia käytetään Scrum-viitekehyksen alustana. Työkalun avulla voidaan suunnitella sprintit ja kehitysjonot etukäteen, ja näiden toteuttamisen edistymistä dokumentoidaan työkalun avulla. Agilefant-alustan käyttö ja Scrum-viitekehyksessä pysyminen on osa kurssin arvostelua, ja tämän takia kurssihenkilökunta käy läpi projektinhallintadataa.

Agilefantin projektinhallintatietomuoto perustuu erilaisiin kehitysjonoihin (backlog, vrt. Scrumin kehitysjono), käyttäjätarinoihin (story) ja tehtäviin (task), ja iteratiiviseen tuotekehitysmalliin. Kehitysjonoja on kolmenlaisia: tuotekehitysjono, projektikehitysjono ja iteraatiokehitysjono. Tuotekehitysjono sisältää kaikki tuotteen projektikehitysjonot ja projektikehitysjonot sisältävät projektin iteraatiokehitysjonot. [20] Scrumia toteutettaessa yksi Scrum sprintti on Agilefantissa yksi iteraatiokehitysjono, joka sisältää sprintin aikana suoritettavat tarinat ja tehtävät.

Käyttäjätarina on osa tuotteen toteutusta, joka toteutuessaan tuo lisäarvoa tuotteelle. Käyttäjätarinalla voi olla useita eri tiloja, esimerkiksi ei aloitettu, tekeillä, odottaa, estetty ja valmis, joiden mukaan tarinoiden etenemistä voidaan seurata. Käyttäjätarinoille voidaan myös asettaa arvo, joiden mukaan tarinoita voidaan arvottaa tärkeyden mukaan. Tehtävä on jokin tarkkaan määritelty ja konkreettinen asia, joka pitää toteuttaa. Käyttäjätarina voi sisältää useamman tehtävän, jotka yhdessä toteutettuna toteuttavat tarinan määrittämän toiminnallisuuden. [25]

4.4 Agilefantin ohjelmointirajapinta

Agilefant tarjoaa datan koneelliseen käsittelyyn REST-rajapinnan, josta projektin dataa voidaan lukea ja muokata. Datan lukemisen ja muokkaamisen yhteydessä tiedonsiirrossa käytetään JSON-tiedostomuotoa. Lyhyt kuvaus rajapinnan tarjoamasta sisällöstä on esitelty taulukossa 1.

Taulukko 1: Agilefantin ohjelmointirajapinnan tarjoamat dataelementit

Elementti	Selite
Attachment	Liitetiedosto
Backlog	Kehitysjono, joka sisältää storyja. Esim. projekti, tuote tai iteraatio.
Comment	Storyyn liittyvä kommentti.
Label	Tagi, jolla voidaan merkata ja jäsentää storyja
Story	Toteutettava ominaisuus tms.
Task	Toteutettava tehtävä
Team	Käyttäjien muodostama joukko
User	Projektin käyttäjä

Rajapinnasta voidaan hakea dataa taulukon 1 dataelementtien mukaan ja vastauksena saadaan listoja näistä elementeistä ja lisäksi niihin liittyvää metadataa. Esimerkiksi käyttäjiä hakiessa voidaan saada lista kaikista projektin käyttäjistä, näiden tunnisteet ohjelmassa, nimet ja sähköpostiosoitteet.

Ohjelmointirajapinnasta tietojen hakeminen onnistuu http-metodeilla seuraavasta osoitteesta:

```
https://app.agilefant.com/<account name>/api/v1/,
```

missä <account name> on käytettävän tunnuksen nimi, joka näkyy verkkosivun URL:ssa Agilefantia verkkoselaimella käytettäessä. Esimerkiksi

```
https://app.agilefant.com/<account name>/api/v1/backlogs/<id>/stories
```

URL:iin tehdyllä http get -kutsulla, saadaan vastauksena yhden työjonon (esimerkiksi koko projektin tai iteraation) kaikki käyttäjätarinat JSON-muodossa, kun <id> on halutun Agilefant-projektin id. Lisäksi URL:iin on lisättävä template, joka määrää kuinka tarkkaan määritellysti data halutaan. Esimerkiksi Update (päivitys) -template antaa datasta kaiken mahdollisen tiedon, ja Summary (yhteenveto) -template tarjoaa vain kompaktimman yhteenvedon datasta. Templatea käyttämättä voidaan rajapinnasta hakea pelkät haettujen dataelementtien tunniste-id:t. Esimerkiksi

```
https://app.agilefant.com/<account name>/api/v1/backlogs/<id>/stories?tem-
plates=Summary
```

haulla voitaisiin saada esimerkiksi seuraavanlainen JSON-vastaus:


```
[ {
  "type" : "story",
  "id" : 2046772,
  "name" : "Gasoline tank",
  "state" : "STARTED",
  "points" : 3,
  "value" : null,
  "dueDate" : null,
  "responsibles" : [ {
    "type" : "user",
    "id" : 196917,
    "initials" : "lassi",
    "fullName" : "lassi",
    "emailMd5" : "b3dd08d4d1552c400661e9e899dd61e9"
  } ],
  "dependsOnStories" : [ ],
  "requiredByStories" : [ ],
  "iterationRank" : {
    "type" : "iterationRank",
    "id" : 2046772,
    "rank" : null
  },
  "effortSpent" : 0,
  "description" : null,
  "labels" : [ ],
  "trelloId" : null,
  "archived" : false,
  "jiraId" : null,
  "hasChildStories" : false,
  "backlogRank" : {
    "type" : "backlogRank",
    "id" : 2046772,
    "rank" : 2
  }
}]
```


Ylläolevassa esimerkissä haetussa työjonossa on ollut vain yksi käyttäjätarina, jonka otsikkona on ”Gasoline tank”, ja joka on tilassa ”aloitettu”. Käyttäjätarinalle on määritetty tärkeysasteeksi 3 ja käyttäjätarinan toteutuksesta vastuussa oleva henkilö on ”lassi”. ”dependsOnStories” listaa käyttäjätarinat, jotka vaaditaan tämän tarinan valmistumiseksi ja ”requiredByStories” listaa käyttäjätarinat jotka ovat riippuvaisia tästä käyttäjätarinasta. Muita käyttäjätarinan määreitä ovat esimerkiski käyttäjätarinaan kulutettu työaika (effortSpent) ja käyttäjätarinaa kategorisoivat tagit (labels). Myös esimerkiksi käyttäjätarinaan liittyvä Jira-ticketin id tai Trello-kortin id ovat mukana tiedoissa, jos tällaiset on käyttäjätarinalle määritetty.

Tarkemmalla templatella (esimerkiksi Update), saataisiin samasta URL:sta lisäksi tietää esimerkiksi mihin työjonoon käyttäjätarina kuuluu, aikamerkinnät käyttäjätarinan luonnille ja sen tilanmuutoksille ja mitä tehtäviä käyttäjätarinaan on liitetty. Käyttäjätarinan

relaatioiden avulla saadaan tiedoksi esimerkiksi siihen liittyvien tehtävien, työjonojen ja muiden käyttäjätarinoiden tunnisteet, joiden avulla taas voidaan hakea lisää tietoa näistä rajapinnan avulla. [26]

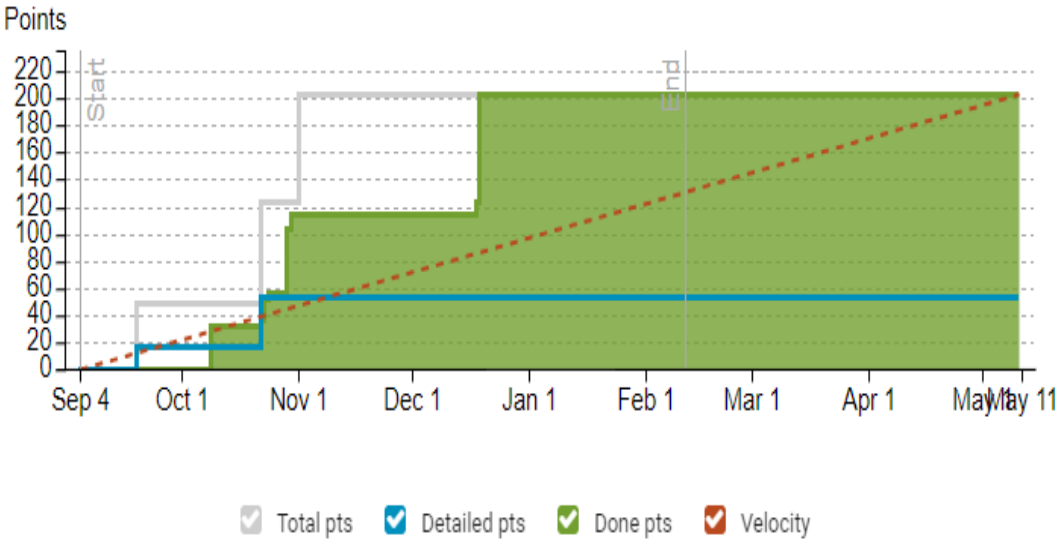
5. VISUALISOINTITYÖKALUN LAAJENNUKSEN TOTEUTUS

Tarkoituksen on toteuttaa n4s-visuun uusi visualisointi, jolla voidaan tarkastella Agilefant-repositorion dataa niin, että mahdolliset poikkeavuudet tai heikkoudet ohjelmistoprojektin prosessien toteutuksessa voitaisiin huomata helposti.

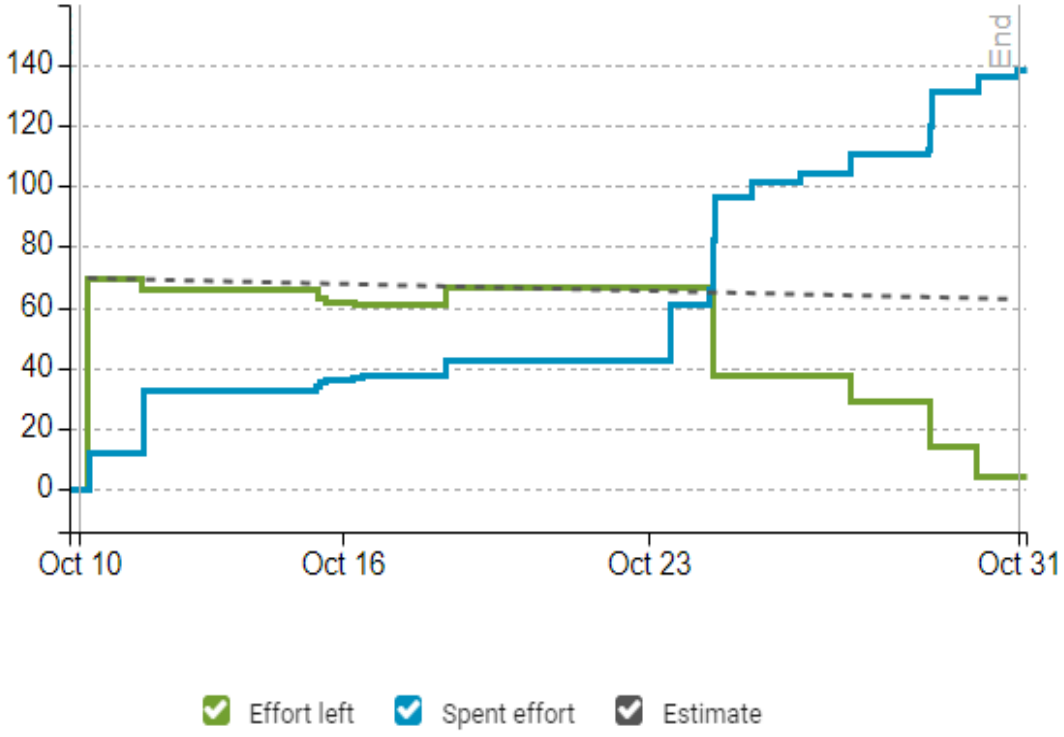
5.1 Suunnitelma

Suunnitelmana on tehdä n4s-visu työkaluun uusi datankerääjämoduuli ja uudesta datasta muodostaa työkalun avulla visualisaatio, jonka avulla voidaan tarkastella Agilefant-projektin etenemistä. Datankerääjä toteutetaan kuvan 7 ”Github parser/linker”, ”Usage parser”, ”Jira parser” ja ”Test parser” -moduulien tavoin niin, että se hakee dataa ulkopuolisesta lähteestä ja käyttää hyväkseen työkalun REST-rajapintaa datan tallentamiseen. Ennen datan tallentamista rajapintaan, tulee se kuitenkin muotoilla n4s-visun määrittelemän datamallin (kuva 9) mukaiseksi.

Agilefantin nykyisessä web-käyttöliittymässä visualisointia on käytetty erilaisten listanäkymien esittämisen tueksi lähinnä erilaisissa burnup ja burndown kuvaajissa (kuvat 11 ja 12), joissa on esitetty tehtyjen ja tekemättömien käyttäjätarinoiden määrää niille annettujen pisteiden avulla. Suunnitelmana on laajentaa n4s-visu-työkalua tekemällä tähän uusi datankeruumoduuli, joka hakee Agilefantin REST-rajapinnasta projektin tiedot ja koostaa niistä työkalun käyttämään tietomuotoon dataa visualisoitavaksi.



Kuva 11: Agilefantin projektin burn up -kuvaaja



Kuva 12: Agilefantin iteraation burn down -kuvaaja

Koska projektinhallintatyökalua käytetään Scrum-viitekehyksen mukaisesti, olennaisia tietoja ovat projektin iteraatiot ja näihin iteraatioihin sisälletyt käyttäjätarinat. Näistä tiedoista saadaan n4s-visun datamallin mukaisia luomalla jokaisesta käyttäjätarinasta konstruktio, jolla on tapahtumat (tilanmuutokset) käyttäjätarinan toteutuksen aloittamiselle ja lopettamiselle. Lisäksi tähän konstruktioon voidaan liittää tapahtumiksi esimerkiksi siihen liittyvät kommentit ja tehtävät.

Esimerkiksi n4s-visun Git issue -kuvaajan (kuva 6) kaltainen kuvaaja, jossa esitettäisiin projektin käyttäjätarinoiden etenemistä ajan funktiona, voisi luoda yleiskuvan projektin tilasta ja siitä kuinka hyvin ketterää tuotantomenetelmää on käytetty. Jos esimerkiksi iteraatiot on suunniteltu hyvin ja kaikki tehtävät on saatu suoritettua iteraation aikana, muodostuu kyseisen kaltaiselle kuvaajalle selkeä porrasmainen muoto.

Suunnitelmana oli toteuttaa n4s-visun valmiita näkymiä ja tietomalleja hyödyntäen ohjelmaan datankerääjäfunktio, joka hakisi dataa Agilefantin ohjelmointirajapinnan avulla halutusta repositoriosta ja tallentaisi sen ohjelman käyttämässä muodossa sen tietokantaan. Tietojen haun, muotoilun ja tallentamisen jälkeen dataa voitaisi tarkastella jo ohjelmasta löytyvien visualisointinäkymien avulla.

5.2 Toteutus

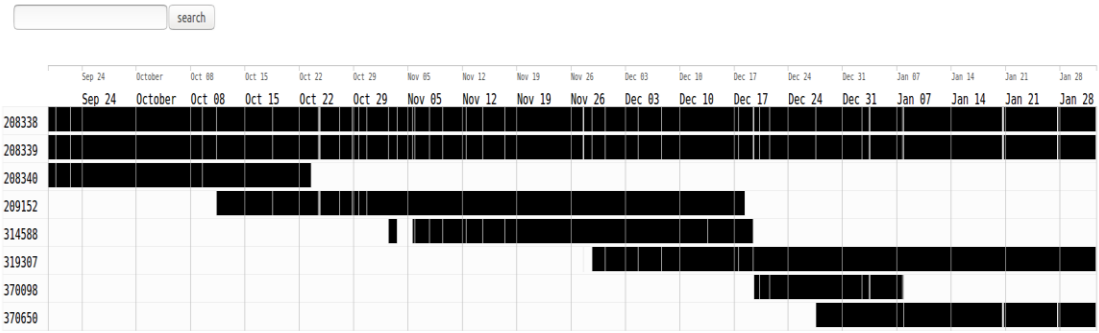
Uutta datankerääjää työkaluun ja visualisaatiota Agilefant-datalle lähdettiin toteuttamaan iteratiivisesti, sillä tarkkaan määriteltyä formaattia lopulliselle visualisaatiolle ei vielä ollut. Tarkoituksena oli aloittaa Agilefantin datan tarkastelusta ja sen tallentamisesta ja tämän jälkeen tutkia erilaisia visualisointimahdollisuuksia n4s-visu työkalun visualisointinäkymien avulla.

Datankerääjän toteutus aloitettiin luomalla ohjelmafunktio, joka hakee Agilefantin ohjelmointirajapinnan avulla halutusta repositoriosta dataa. Rajapinnasta saa yhdellä http-kutsulla koko repositorion kaikki käyttäjätarinat JSON-muodossa ja näistä otetaan talteen halutut tiedot ja pakataan ne Javascript-objektiksi n4s-visun ennalta määritetyn tietokantaskeeman mukaiseen muotoon (kuva 9). Ajan suhteen käyttäjätarinoiden etenemistä kuvaavaa visualisointia varten käyttäjätarinoista otetaan talteen tarinan työstämisen aloitusaika ja ajankohta, jolloin tarina on merkitty valmiiksi. Datankerääjä lähettää tietokantaan käyttäjätarinaobjektin ja tämän jälkeen luo objektiin linkitettävät tilanmuutostapahtumat tarinan alku ja loppupäivämäärille, näin visualisointinäkymä voi kuvata käyttäjätarinan kehityksen elinajan.

N4s-visun datamallin mukaiseksi Agilefantin data saatiin luomalla konstruktioita käyttäjätarinoista ja luomalla näille konstruktioille tilanmuutostapahtumat avattu (opened)

käyttäjätarinan toteutuksen aloittamisen ja suljettu (closed) käyttäjätarinan valmistumiselle. Tämän lisäksi ohjelmointirajapinnasta haetaan käyttäjätarinoiden kommentit ja näistä luodaan tapahtumat. Kaikki tapahtumat linkitetään n4s-visun tietokannassa niihin liittyviin konstruktioihin.

Kun dataa oli onnistuneesti saatu tallennettua Agilefantista työkalun tietokantaan, luotiin datasta visualisaation ensimmäinen iteraatio, joka on kuvattu kuvassa 13. Tässä versiossa sprintit on kuvattu omilla riveillään, niin, että jokaisen sprintin kaikkien tarinoiden toteutusajat on piirretty samalle riville päällekkäin, jolloin kuvaajan alimmille kuudelle riville muodostuu näkyväksi jonkinlainen kuva jokaisen sprintin alkuajasta ja lopusta. Tämän näkymän tarkastelun jälkeen sovittiin kuitenkin, että seuraavaan versioon käyttäjätarinat jaetaan jokainen omille riveilleen, niin ettei päällekkäisyyksiä tapahdu, jolloin jokaisen käyttäjätarinan etenemisen pituutta voidaan tarkastella yksitellen.



Kuva 13: Uuden visualisaation ensimmäinen iteraatio

Kuvassa 14 on esitetty datankerääjäohjelman funktio, joka hakee käyttäjätarinat, muodostaa niistä visualisointialustan datamallin mukaisia objekteja ja lähettää ne alustan tietokantaan. Funktio suorittaa http get -kutsun Agilefantin REST-ohjelmointirajapinnan URL:iin

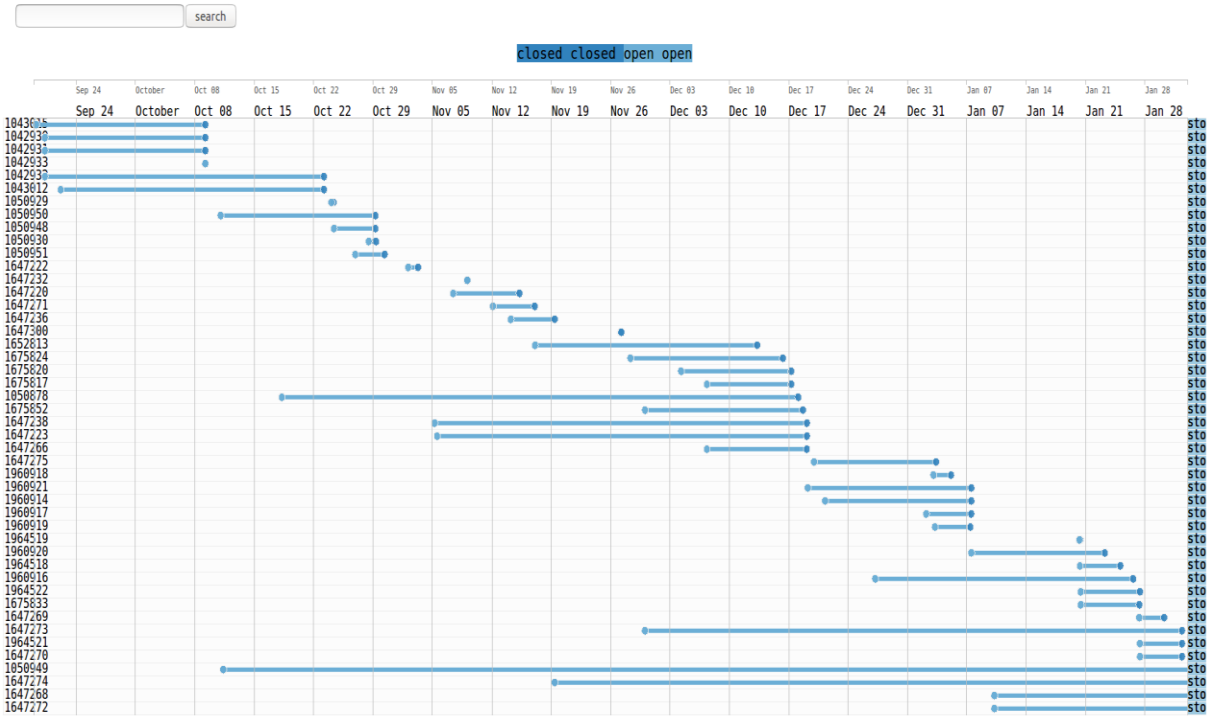
```
/api/v1/backlogs/<projectId>/stories?templates=Update,
```

josta se saa vastauksena halutun projektin kaikki käyttäjätarinat. Käyttäjätarinat käsitellään yksitellen for-loopin avulla ja näistä muodostetaan Javascript-objekteja, joihin määritetään n4s-visun datamallin vaatimat tiedot (aikaleimat, nimi, lähde). Luotu käyttäjätarinaobjekti lähetetään tietokantaan postToDb-funktiokutsulla. Tietokantaan lähettämisen jälkeen kutsutaan funktioita, jotka luovat tilamuutokset käyttäjätarinalle (createStateChangeEvents(newConstruct) rivillä 91 kuvassa 14) ja haetaan käyttäjätarinan mahdolliset kommentit (getComments(newConstruct) rivillä 92 kuvassa 14)

```
function getStories() {
    request(
        {
            url : agileRepo + "/api/v1//backlogs/" + projectId + "/stories" + "?templates=Update",
            headers : {
                "Authorization" : auth
        }
        },
        function (error, response, body) {
            //console.log(error);
            o = JSON.parse(body);
            var item = {};
            for(var i = 0; i < o.length; i++) {

                item = {};
                item = o[i];
                if(item.endDate != null && item.startDate != null && item.iteration != null && item.iteration.name != null) {
                    var constructResult = {};
                    var construct = {};
                    construct.type = item.iteration.name;
                    construct.origin_id = [ { 'source_id': String( item.id ), 'source': agileRepo, context: item.backlog.id } ];
                    construct.name = item.name;
                    var meta = {};
                    meta.startTime = new Date(item.startDate);
                    meta.endTime = new Date(item.endDate);
                    //add 1 second to endTime so the visualizator doesn't spazz out if the endtime is the same as the start
                    meta.endTime.setSeconds(meta.endTime.getSeconds() + 10);
                    meta.id = item.id;
                    construct.data = meta;


                    postToDb(constructsApi, construct, constructResult, constructResult.eventIds, function ( newConstruct ) {

                        createStateChangeEvents(newConstruct);
                        getComments(newConstruct);
                    });
                }
            }
        }
    );
}
```

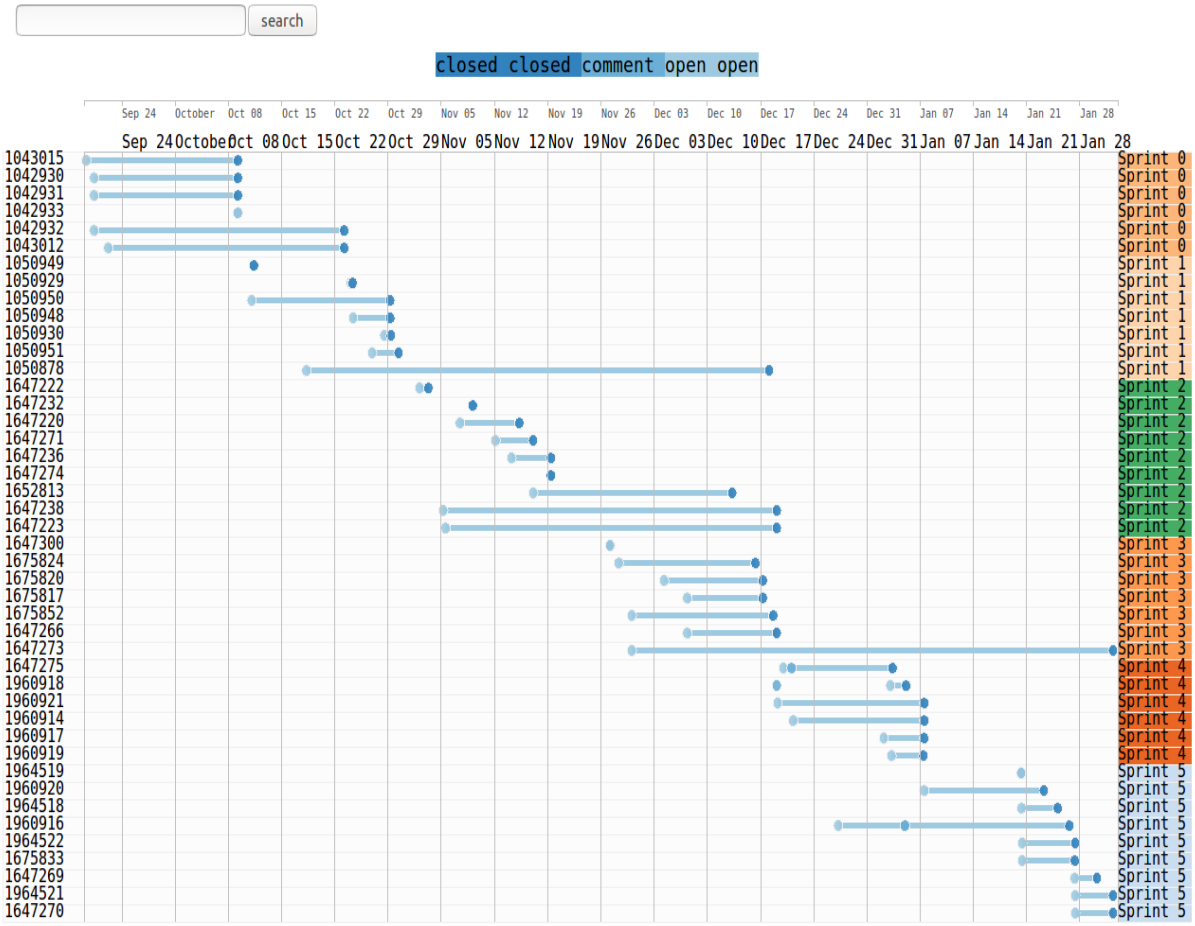
Kuva 14: Funktio, joka hakee käyttäjätarinat Agilefantin REST-rajapinnasta ja lähettää ne työkalun tietokantaan

Kuvassa 15 on esitetty kuvakaappaus visualisaationäkymän toisesta iteraatiosta. Tässä käyttäjätarinoiden pituudet on esitetty tarinan toteuttamisen aloitusajasta sen lopetusaikaan asti. Tähän iteraatioon Agilefant-datan tietomallia muokattiin niin, että saatiin jo aiemmin n4s-visuun toteutettua issue-näkymää (kuva 5) hyödynnettyä suoraan uudella datalla. Tässä vaiheessa kaikki käyttäjätarinat on järjestetty ajallisesti aloitus- ja lopetusajankohtien mukaan, eikä täyttä kuvaa iteraatioiden kokonaisuuksista saa vielä. Tässä vaiheessa päätettiin, että seuraavaan versioon lisätään mukaan näkyviin iteraatiot ja kommentit.



Kuva 15: Uuden visualisaation toinen iteraatio

Visualisaation kolmas iteraatio on esitetty kuvassa 16. Käyttäjätarinat on nyt järjestetty iteraatioiden mukaan, kun edellisessä versiossa järjestys oli puhtaasti tarinoiden toteutusajan mukaan, joka aiheutti vääristyneen kuvan projektin etenemisestä. Riveille on nyt merkattu oikeaan laitaan mihin iteraatioon käyttäjätarina on kuulunut, jolloin voidaan nähdä suoraan, jos jonkin tarinan toteutus on venynyt selvästi yli muun iteraation toteutuksen, kuten on käynyt kuvan esimerkkiprojektissa sprinteissä 1 ja 3 olevien yksittäisten käyttäjätarinoiden osalta. Lisäksi tässä versiossa on merkattu käyttäjätarinoiden kommentit kuvaajalle.



Kuva 16: Uuden visualisaation kolmas iteraatio

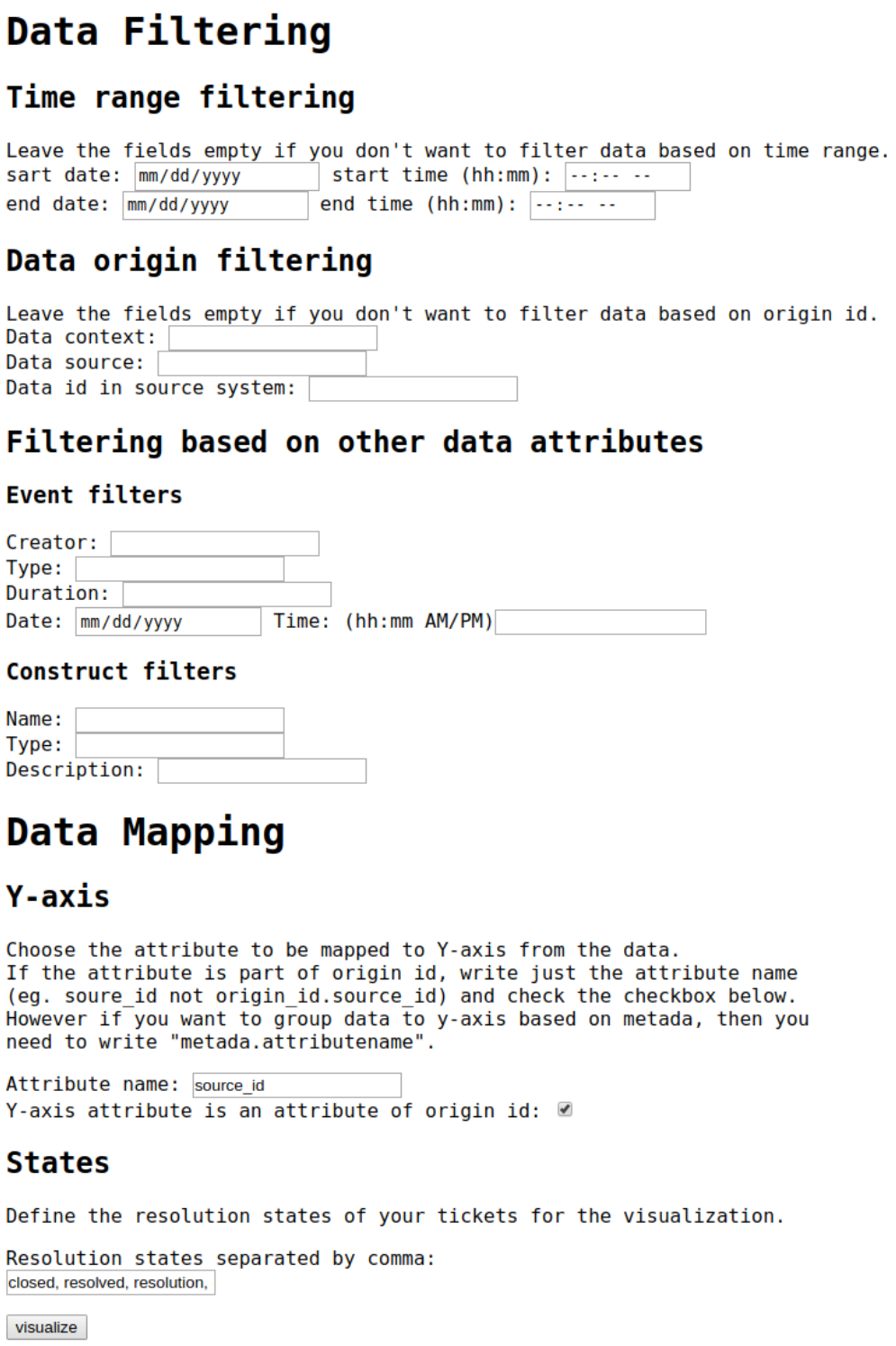
5.3 Uuden työkalun käyttö

Toteutetun uuden datankerääjän käyttö onnistuu ajamalla datankerääjäohjelmaa agilefantparser.js parametreillä Agilefant-repositorion url-osoitteella, projektin id:llä, käyttäjätunnuksen sähköpostiosoitteella ja salasanalla:

```
node agilefantparser.js "repo_url" "project id" "email" "password"
```

Tällöin datankerääjä hakee repositoriosta annetun projektin kaikki käyttäjätarinat ja näihin liittyvät kommentit ja muodostaa käyttäjätarinosita datamallin mukaisia konstruktioita. Lisäksi käyttäjätarinoiden toteutuksen aloittamiselle ja lopettamiselle muodostetaan tapahtumaobjektit ja nämä linkitetään käyttäjätarinan konstruktioon. Myös kommenteista tehdään tapahtumaobjekteja, jotka linkitetään kommenttiin liittyvän käyttäjätarinan konstruktioon. Objektit tallennetaan niiden luonnin jälkeen n4s-visun tietokantaan REST-rajapinnan avulla.

Kun halutun projektin datat on haettu datankerääjätyökalulla, voidaan visualisointityökalulla muodostaa datasta visualisaatio. Agilefant-datan visualisointi onnistuu alustan issue-timeline -visualisaation avulla. Visualisaatiota muodostaessa siihen voidaan rajata tietokannasta haluttu data. Työkalun datansuodatusnäkymä on esitetty kuvassa 17. Uuden Agilefant-datan tapauksessa ”Data origin filtering”-kohdan ”Data context”-kenttään voidaan laittaa Agilefant-projektin id, jolloin saadaan näkyviin tietyn projektin visualisaatio. Muiden suodatusvaihtoehtojen avulla visualisaatiota voidaan rajata esimerkiksi tietylle aikavälille, jolloin halutessa dataa voidaan visualisoida vain yksi sprintti kerrallaan.

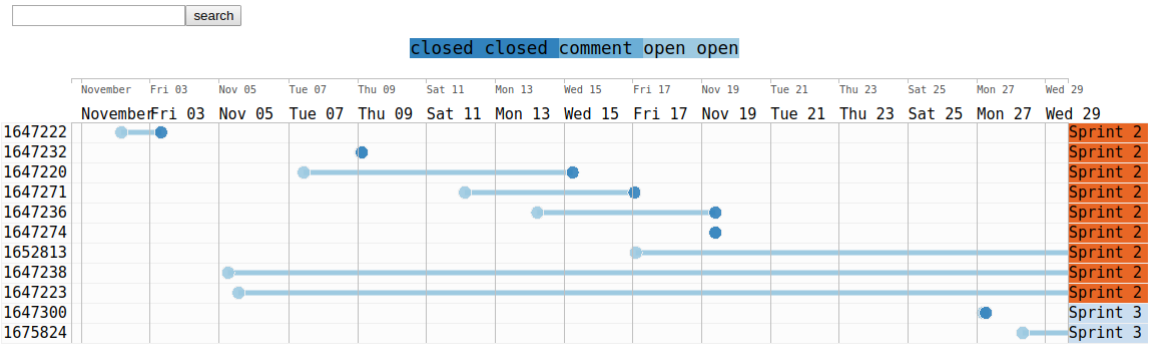


Kuva 17: Visualisointityökalun datansuodatusnäkymä

Muita suodatusmahdollisuuksia työkalussa on esimerkiksi datamallin konstruktioiden (Construct filters) tai tapahtumien (Event filters) mukaan rajaaminen. Tapahtumien mu-

kaan rajatessa on esimerkiksi mahdollista ottaa visualisaatioon mukaan pelkästään tapahtumat, joiden tyyppi on ”suljettu”, jolloin nähdään pelkästään käyttäjätarinoiden valmistumisajat.

Kuvassa 18 on esitetty luvussa 4.1.2 käytetyn esimerkkiprojektin visualisaatio rajattuna vain yhden kuukauden ajalle. Tällöin voidaan visualisaatiossa nähdä tarkemmin tarkat aikaleimat ja esimerkiksi käyttäjätarinoiden valmistumisen tarkat päivämäärät.



Kuva 18: Esimerkkiprojektin eteneminen rajattuna marraskuulle

6. TYÖKALUN JAKAMINEN JA KÄYTTÖ

Modernien verkkoapplikaatioympäristöjen käyttämät riippuvuudet muuttuvat niin kiivaaseen tahtiin, että n4s-visu työkalun asennuskaan ei enää ole onnistunut täysin suoraviivaisesti sille kirjoitettujen asennusohjeiden mukaisesti. Tässä luvussa on tarkasteltu mahdollisuuksia työkalun saatavuuden parantamiseksi.

Työkalun tulevaa jakamista ja käyttöä varten harkittiin useaa eri vaihtoehtoa. Varteenotettavimmat vaihtoehdot olivat virtuaalikoneympäristön käyttö, työkalun asennusohjeiden päivitys toistettavaksi ja luotettavammaksi prosessiksi ja työkalun palvelinasennus pilviympäristöön.

Ensimmäinen idea oli, että luotaisiin valmiiksi asennettu ja konfiguroitu virtuaalikoneympäristö, jossa työkalu olisi asennettuna ja se toimisi suoraan virtuaalikoneessa ilman, että käyttäjän tulisi itse asentaa mitään. Virtuaalikoneeseen asennettaisiin Linux-pohjainen käyttöjärjestelmä ja tälle alustalle asennettaisiin työkalun tarvittavat riippuvuudet ja itse työkalu valmiiksi. Virtuaalikoneympäristön luominen on melko yksinkertaista, mutta jakamista saattaa vaikeuttaa virtuaalikonetiedostojen suuri koko. Esimerkiksi valmiiksi asennettu Ubuntu-virtuaalikonetiedosto vie tilaa viidestä kymmeneen gigatavuun. Tämän takia onkin syytä tarkastella myös muita vaihtoehtoja.

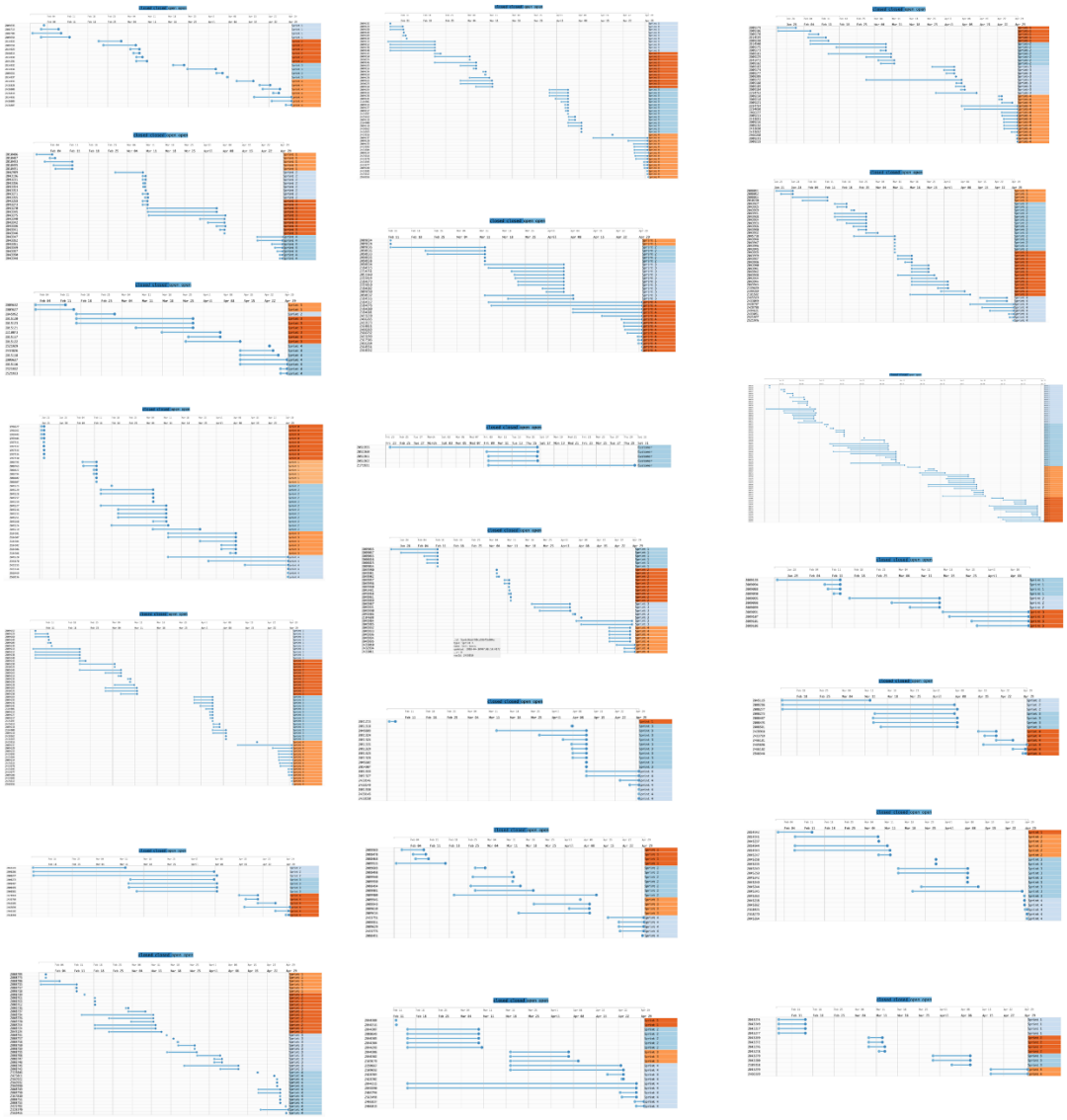
N4s-visu työkalun asennusohjeita ei ole päivitetty vuosiin ja voi olla mahdollista, että ohjeita päivittämällä saadaan käyttöön yksinkertainen ja toistettava prosessi, jolla työkalu saadaan asennettua luotettavasti. Vahvuutena tässä vaihtoehdossa on se, että työkalua voitaisiin jakaa yksinkertaisesti palvelimella olevalla Git repositoriolla, josta jokainen käyttäjä voisi kloonata repositorion ja asentaa työkalun omalle työasemalleen. Tämä ei tuottaisi merkittävästi lisätyötä ja loppukäyttäjä saisi myös mahdollisuuden ympäristön täyteen hallintaan.

Työkalusta voitaisi myös toteuttaa versio, joka asennettaisiin esimerkiksi yliopiston palvelimelle, johon pääsisi käsiksi esimerkiksi sisäverkosta. Tämä vaihtoehto olisi kaikista käyttäjäystävällisin loppukäyttäjälle, mutta vaatisi panostusta ylläpitoon. Visualisointialustan datankerääjäohjelmat toimivat komentoriviltä, joten aivan täyttä verkkopohjaista alustaa ei tällaisenaan saataisi aikaan, mutta visualisaatioiden tarkastelu onnistuisi verkkoselaimella palvelimelta. Datankerääjäohjelmille omat näkymät asiakasohjelmaan toteuttamalla saataisiin koko n4s-visu alusta toimimaan palvelinpohjaisena verkkopalveluna.

Visualisointityökalun laajennuksen kehityksen jälkeen saatiin alkuperäistä asennusohjetta muokattua sellaiseen muotoon, että ohjelman asennus lähdekoodista onnistuu toistettavasti. Ohjelma saatiin palvelimelle asennettua niin, että kurssihenkilökunta pääsee käsiksi kurssidatan visualisaatioihin.

7. TULOKSET

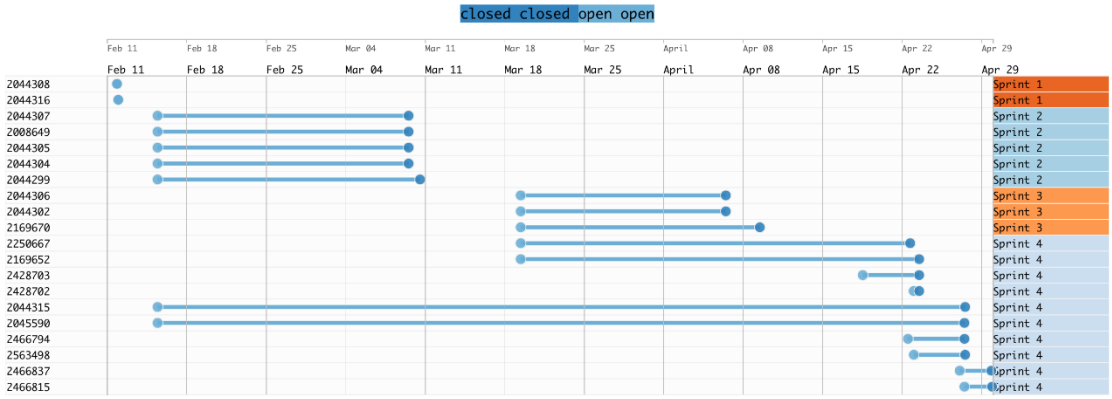
N4s-visun uuden datankerääjälisäosan kehityksen tuotteena saatiin visualisointialustalle uusi visualisointikohde Agilefantin projektinhallintadatasta ja uusi visualisaatio, josta nähdään projektin eteneminen sprinteittäin. Datankerääjän valmistumisen jälkeen ohjelma saatiin asennettua yliopiston palvelimelle ja sitä päästiin kokeilemaan oikealla käyttäjädatalla Software Engineering Methodologies opintojakson harjoitustöiden arvostelun yhteydessä. Kuvassa 19 on esitetty kahdenkymmenenyhden eri harjoitustyöryhmän projektien kuvaajat.



Kuva 19: Software Engineering Methodologies opintojakson harjoitustyöryhmien Agilefant-datan visualisaatioita

Kun opintojakson harjoitustyöryhmien projektidatat oli saatu ajettua ohjelmaan, käytiin kuvaajat läpi sellaisenaan kurssihenkilökunnan kesken. Kurssiassistentteja pyydettiin koostamaan lyhyt mielipide projektin etenemisestä ja Scrum-viitekehyksen noudattamisesta visualisaation perusteella. Vastaukset on esitetty kokonaisuudessaan visualisaatioiden kera liitteessä B.

Kurssihenkilökunnan analyysissa huomioitiin erityisesti käyttäjätarinoiden toteutuksen pysyminen sprintin rajojen sisällä. Kuvaajasta näkeekin hyvin, jos jonkin tarinan suoritus on venynyt seuraavan sprintin toteutuksen puolelle. Esimerkiksi Kuvan 20 visualisaatiossa on näkyvillä tilanne, jossa kaksi käyttäjätarinaa on avattu jo toisen sprintin aikana, mutta ne on saatu valmiiksi vasta neljännessä sprintissä.



Kuva 20: Erään harjoitustyöryhmän projektin etenemisen visualisaatio

Yleisiä mielipiteitä työkalun toiminnasta opintojakson arvioinnin tukena kysyttäessä saatiin positiivista palautetta. Assistenttien mukaan visualisaatiosta näkee nopeasti, jos projektin etenemisessä tapahtuu jotain poikkeavaa ja kuvaajat ovat hyödyllisiä projektien vertailuun keskenään. Visualisaatiosta näkee myös, miten ryhmä jakaa työmäärän projektin ajalle, ja kuinka hyvin sprinttien määrittämässä aikataulussa pysytään.

Agilefantin jo tarjoamien burn up- ja burn down-kuvaajien esittämien työmäärien toteutumisten lisäksi uusi visualisaatio tarjoaa näkymän, jossa käyttäjätarinat on eritelty niin, että voidaan tarkastella pelkkien työmäärien sisään selkeitä kokonaisuuksia prosessin etenemisessä. Agilefantin omien työmäärien kuvaajia voidaankin mahdollisesti käyttää yhdessä uuden visualisaation kanssa, jolloin voidaan vertailla esimerkiksi uudessa visualisaatiossa näkyvää käyttäjätarinoiden määrää Agilefantin kuvaajan työmääriin. Jatkokehitysideana nämä työmäärät voisi jotenkin yrittää saada mukaan n4s-visun visualisaatioon,

jolloin käyttäjätarinoiden toteutuksen pituutta ja niille arvioitua työmäärää voitaisiin vertailla keskenään.

8. YHTEENVETO

Tässä diplomityössä lähdettiin selvittämään, miten ohjelmistokehityksen projektinhallintadataa voitaisiin visualisoida ketterän ohjelmistoprojektin etenemisen seuraamisen helpottamiseksi. Selvitystä lähdettiin tekemään aiheen kirjallisuuskatsauksen ja tapaustutkimuksen avulla.

Kirjallisuuskatsauksessa tutkittiin ketterän ohjelmistokehityksen periaatteita ja menetelmiä, ja ohjelmistojen visualisointia. Ketterän kehityksen työkaluista paneuduttiin erityisesti Scrum-viitekehykseen, jota käytetään nykyään laajalti modernissa ohjelmistokehityksessä ja myös toteutetun visualisointiohjelman laajennuksen datalähteen käytössä. Myös ohjelmistotuotantoprosessissa käytettäviä ohjelmallisia projektinhallintaa helpottavia alustoja esiteltiin. Ohjelmistojen visualisoinnin kirjallisuuteen liittyen materiaalia itse ohjelmistontuotantoprosessin visualisoinnista löytyi melko vähän ja etenkin prosessin retrospektiivisen visualisoinnin saralta työtä on tehty vähän.

Tampereen Teknillisellä Yliopistolla kehitetty projektinhallintadatan visualisointialusta esiteltiin ja alustan käyttöä, toimintaa ja teknisiä ominaisuuksia kuvattiin. Opetustyössä käytetty projektinhallinta-alusta esiteltiin ja tämän alustan ohjelmointirajapinnan rakennetta ja käyttöä tutkittiin. Määriteltiin visualisointialustalle haluttavan uuden datankerääjätyökalun tarve.

Tapaustutkimuksessa toteutettiin jo olemassa olevalle visualisointialustalle uusi datankerääjäohjelma ja visualisaatio, jonka avulla voidaan tarkastella ohjelmistoprojektin etenemistä sen toteutuksen elinkaaren aikana. Kehitystyön vaiheet on dokumentoitu ja kehityksen etenemisen aikana tehdyt valinnat ja käytetyt ratkaisut on selvitetty. Uuden datankerääjäohjelman käyttöä ja uuden datalähteen tietojen visualisointia on kuvattu.

Toteutettua työkalua on kokeiltu ohjelmistotuotannon menetelmien arvioinnin tukena yliopiston opintojaksolla. Ohjelman laajennus onnistui, uuden datalähteen datan kerääminen toimii ja data saadaan tallennettua ohjelman tietokantaan sellaiseen muotoon, että visualisointiohjelmalla saadaan piirrettyä visualisaatio datasta. Ohjelmaa on testattu yliopiston opintojakson arvioinnin tukena ja sitä testanneen kurssihenkilökunnan mukaan ohjelman tuottamalla visualisaatiolla saadaan uusi näkökulma ohjelmistoprojektin etenemiseen.

Jatkokehitysideana n4s-visun Agilefant-visualisaatioon voisi ottaa mukaan Agilefantin ohjelmointirajapinnan tarjoamia tietoja, joita ei visualisaatiossa tällä hetkellä käytetä. Esimerkiksi mahdolliset liitetiedostot, käyttäjätarinoiden luokitteluelementit (Label) ja käyt-

täjätiedot voitaisiin tuoda mukaan visualisaatioon. Lisäksi käyttäjätarinoiden datasta voitaisiin ottaa mukaan myös mahdolliset käyttäjätarinalle määritetyt arvioidut ja toteutuneet työmäärät.

LÄHTEET

[1] Kaushal Chari, Manish Agrawal (2018), Impact of incorrect and new requirements on waterfall software project outcomes, Empirical Software Engineering February 2018, Volume 23, Issue 1, pp 165–185

[2] Mattila, A-L, Systä, K, Sievi-Korte, O, Leppänen, M & Mikkonen, T (2017), Discovering Software Process Deviations Using Visualizations. in Agile Processes in Software Engineering and Extreme Programming: 18th International Conference, XP 2017, Cologne, Germany, May 22-26, 2017, Proceedings. Lecture Notes in Business Information Processing, vol. 283, Springer, pp. 259-266, International Conference on Agile Software Development

[3] Mattila, A-L, Luoto, A, Terho, H, Hylli, J, Sievi-Korte, O & Systä, K (2015), Unified Model for Software Engineering Data. in 2015 IEEE 3rd Working Conference on Software Visualization (VISSOFT). IEEE, Bremen, Germany, pp. 150-154, IEEE Working Conference on Software Visualization, 1/01/00.

[4] Manifesto for Agile Software Development, http://agilemanifesto.org/, viitattu 2018

[5] Kisielnicki, J., & Misiak, A. M. (2017). EFFECTIVENESS OF AGILE COMPARED TO WATERFALL IMPLEMENTATION METHODS IN IT PROJECTS: ANALYSIS BASED ON BUSINESS INTELLIGENCE PROJECTS. Foundations of Management, 9(1), 273-286.

[6] Vuorinen, Jyri, (2011). Scrum-menetelmän käyttö Pirkanmaalaisissa ohjelmistoyrityksissä, Tampereen teknillinen yliopisto Saatavissa: URN:NBN:fi:tty-201102171040

[7] Ken Schwaber and Jeff Sutherland, (2017) The Scrum Guide, saatavilla: https://www.scrumalliance.org/learn-about-scrum/the-scrum-guide

[8] Muhammad Ovais Ahmad, Denis Dennehy, Kieran Conboy, Markku Oivo (2018), Kanban in software engineering: A systematic mapping study, Journal of Systems and Software, Volume 137, March 2018, pp 96-113

[9] HowardLei, Farnaz Ganjeizadeh, Pradeep Kumar Jayachandran, Pinar Ozcan (2017), A statistical analysis of the effects of Scrum and Kanban on software development projects, Robotics and Computer-Integrated Manufacturing Volume 43, February 2017, pp 59-67

[10] Trello, https://trello.com, viitattu 2018

[11] Jira | Issue & Project Tracking Software https://www.atlassian.com/Jira, viitattu 2018

[12] Spinellis, Diomidis. (2005), Version Control Systems IEEE Software; Los Alamitos Vol. 22, Issue 5, (Sep/Oct 2005): pp. 108-109.

[13] Krill, Paul. (2011), Torvalds's Git: The 'it' technology for software version control InfoWorld.com; San Mateo (Jul 26, 2011).

[14] GitHub, https://github.com, viitattu 2018

[15] Meyer, Mathias, (2014), Continuous Integration and Its Tools, IEEE Software, Volume: 31, Issue: 3, May-June 2014, pp. 14-16

[16] Chen, Hsuanwei Michelle, (2017), An Overview of Information Visualization, Library Technology Reports. Vol. 53 Issue 3

[17] Aigner, Wolfgang; Miksch, Silvia; Schumann, Heidrun; Tominski, Christian (2011) Visualization of Time-Oriented Data, Springer-Verlag London Limited, 2011

[18] Riccardo Mazza, Introduction to Information Visualization, (2009), Springer-Verlag London Limited, 2009

[19] Mattila, A-L, Ihantola, P, Kilamo, T, Luoto, A, Nurminen, M & Väätäjä, H (2016), Software visualization today - Systematic literature review. in AcademicMindtrek 2016 - Proceedings of the 20th International Academic Mindtrek Conference. ACM, pp. 262-271, Mindtrek Conference, 1/01/00.

[20] Stephan Diehl, (2017), Software Visualization Visualizing the Structure, Behaviour, and Evolution of Software, Springer-Verlag Berlin Heidelberg, 2007

[21] J. Paredes, C. Anslow, and F. Maurer (2014), Information visualization for agile software development, in Software Visualization (VISSOFT), 2014 Second IEEE Working Conference on. IEEE, 2014, pp. 157–166.

[22] Brandt, Sebastian; Striewe, Michael; Beck, Fabian; Goedicke, Michael (2017), A Dashboard for Visualizing Software Engineering Processes Based on ESSENCE, Software Visualization (VISSOFT), 2017 IEEE Working Conference on. IEEE, 2017

[23] Valerio Cosentino, Javier Luis Cánovas Izquierdo, Jordi Cabota, (2018), Gitana: A software project inspector, Science of Computer Programming Volume 153, 15 February 2018, pp 30-33

[24] need4speed vis tool, https://bitbucket.org/rimina/n4s-visu/, viitattu 2018

[25] Agilefant user guide, https://www.agilefant.com/support/user-guide/, viitattu 2018

[26] Agilefant API https://www.agilefant.com/agilefant-api/, viitattu 2018

LIITE A: KETTERÄN KEHITYKSEN JULISTUKSEN PERIAATTEET

Julistuksen takana olevat periaatteet

Noudatamme seuraavia periaatteita:
Tärkein tavoitteemme on tyydyttää asiakas toimittamalla tämän tarpeet täyttäviä versioita ohjelmistosta aikaisessa vaiheessa ja säännöllisesti.

Otamme vastaan muuttuvat vaatimukset myös kehityksen myöhäisessä vaiheessa. Ketterät menetelmät hyödyntävät muutosta asiakkaan kilpailukyvyn edistämiseksi.

Toimitamme versioita toimivasta ohjelmistosta säännöllisesti, parin viikon tai kuukauden välein, ja suosimme lyhyempää aikaväliä.

Liiketoiminnan edustajien ja ohjelmistokehittäjien tulee työskennellä yhdessä päivittäin koko projektin ajan.

Rakennamme projektit motivoituneiden yksilöiden ympärille. Annamme heille puitteet ja tuen, jonka he tarvitsevat ja luotamme siihen, että he saavat työn tehtyä.

Tehokkain ja toimivin tapa tiedon välittämiseksi kehitystiimille ja tiimin jäsenten kesken on kasvokkain käytävä keskustelu.

Toimiva ohjelmisto on edistymisen ensisijainen mittari.

Ketterät menetelmät kannustavat kestävään toimintatapaan. Hankkeen omistajien, kehittäjien ja ohjelmiston käyttäjien tulisi pystyä ylläpitämään työtahtinsa hamaan tulevaisuuteen.

Teknisen laadun ja ohjelmiston hyvän rakenteen jatkuva huomiointi edesauttaa ketteryyttä.

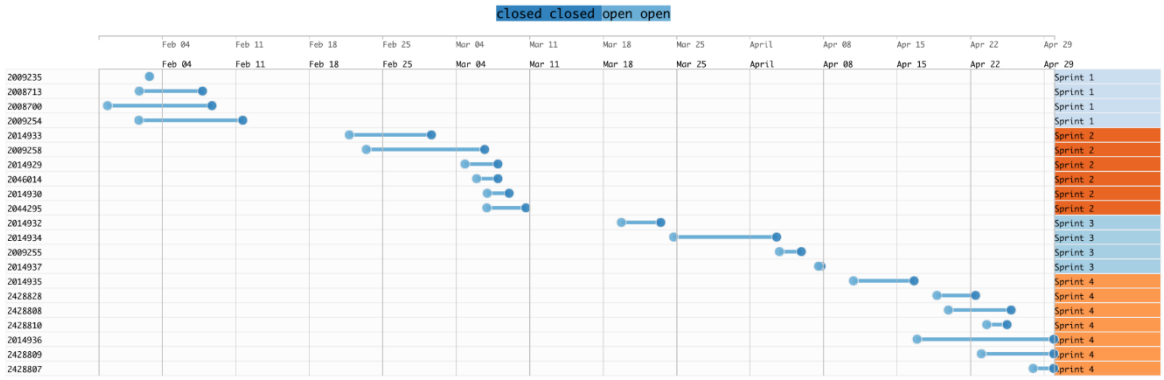
Yksinkertaisuus - tekemättä jätettävän työn maksimointi - on oleellista.

Parhaat arkkitehtuurit, vaatimukset ja suunnitelmat syntyvät itseorganisoituvissa tiimeissä.

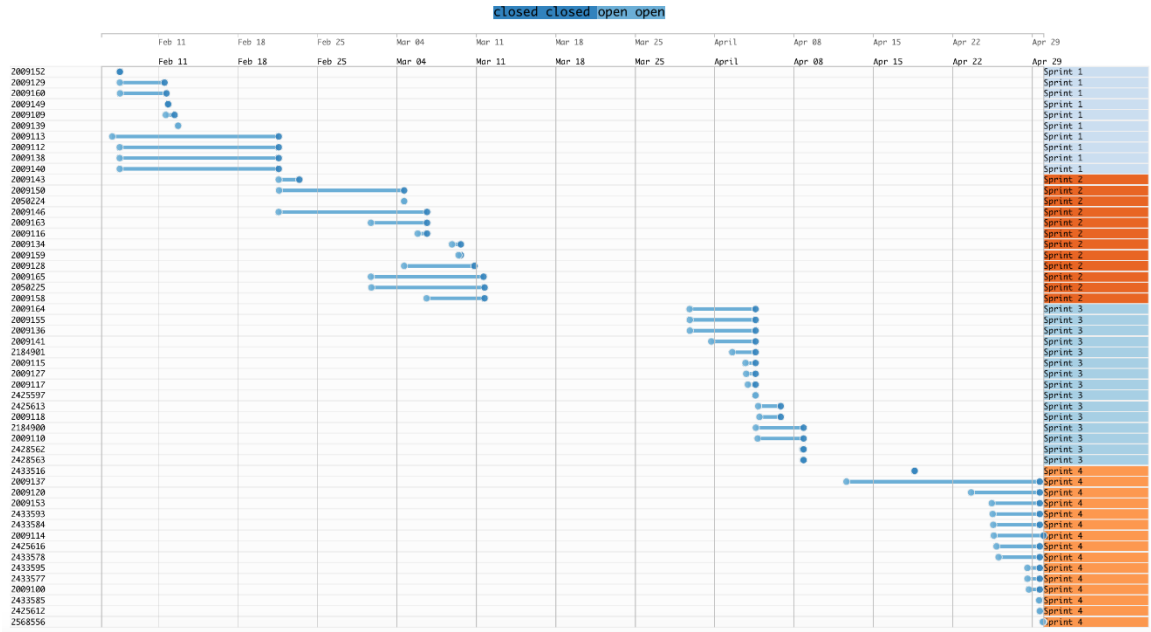
Tiimi tarkastelee säännöllisesti, kuinka parantaa tehokkuuttaan, ja mukauttaa toimintaansa sen mukaisesti.

LIITE B: KURSSIHENKILÖKUNNAN SUORITTAMA AGILEFANT VISUALISAATIOIDEN ANALYYSI

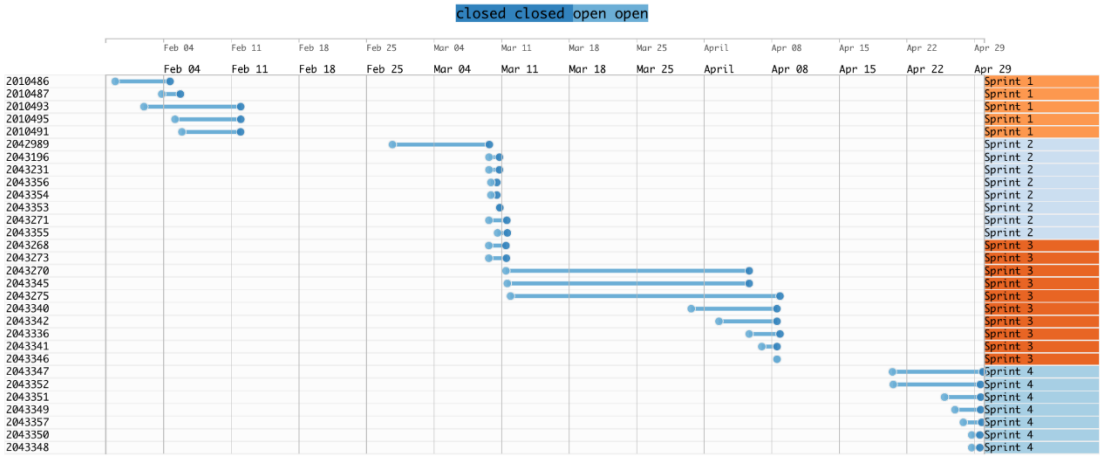
00: The group might not have much structure in the process: the stories have been always opened and closed individually. However, they have divided the stories considering the sprints so that they do not have any stories opened in one sprint and closed in another.



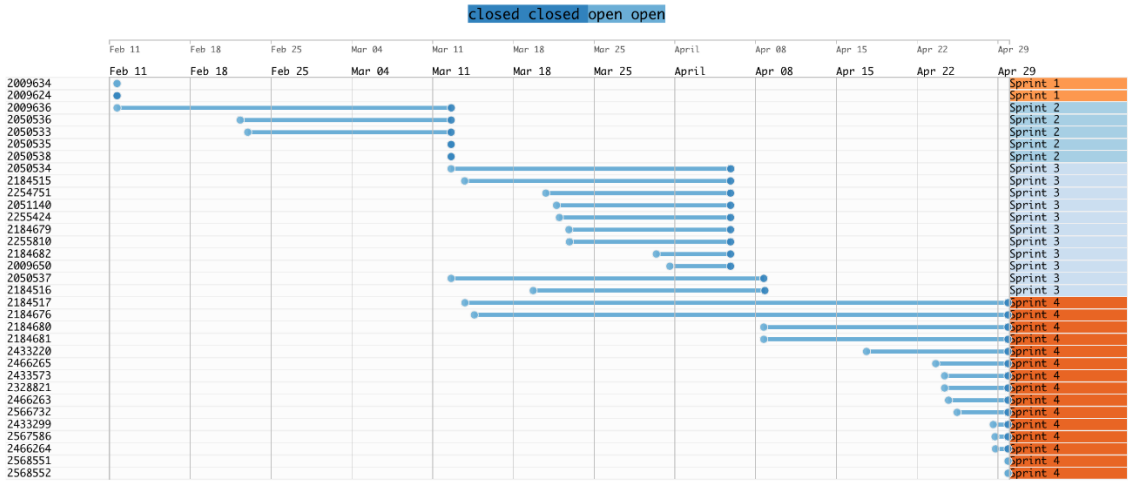
01: The sprints and their deadlines stand out quite clearly. The group has used some kind of a standard (meeting?) on when to close stories. Stories have been opened individually.



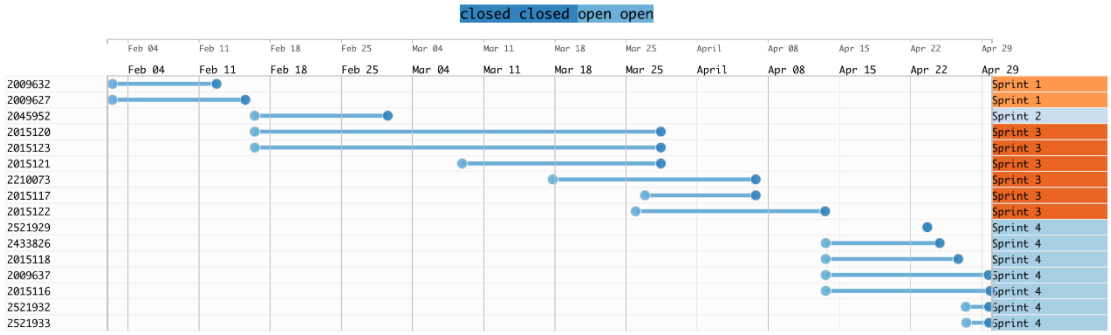
02: The group has probably started to use Agilefant only in the end of sprint 2 / beginning of sprint 3.



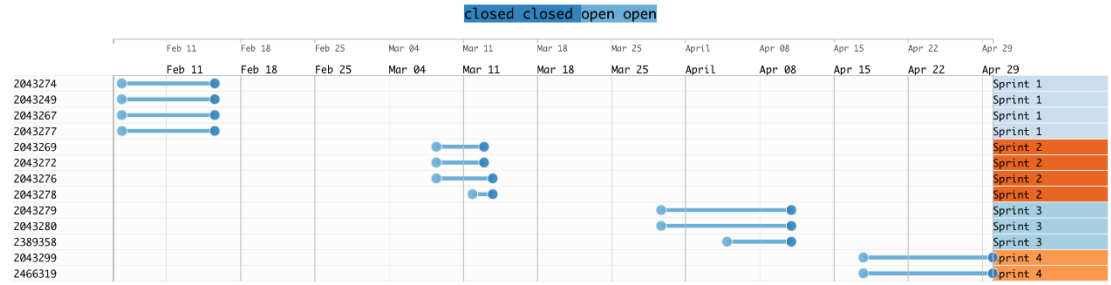
03: No use of agilefant in sprint 1, but very structured process in the rest of the sprints (all stories from the sprint closed in one time). Sprint 3 has had too much (too difficult?) work, and some stories have been continued in sprint 4.



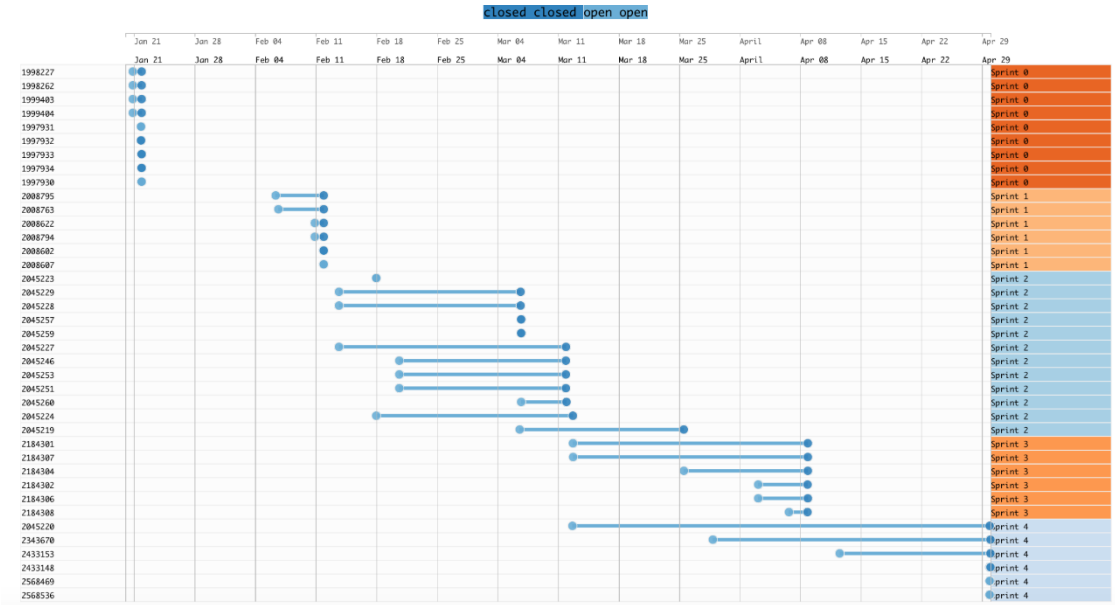
04: Basically no use of agilefant in sprints 1 and 2.



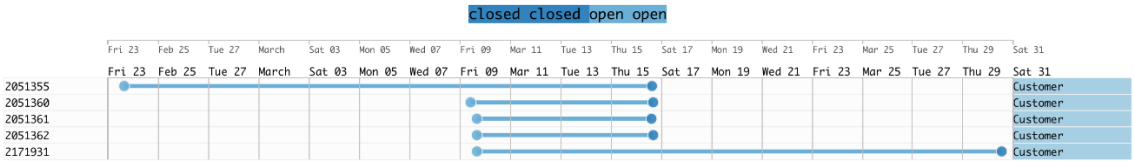
05: Very structured use of Agilefant. The team have probably had clear meetings where stories have been opened and closed together. The size of a story has been estimated well, since each story has been worked on pretty equal time.



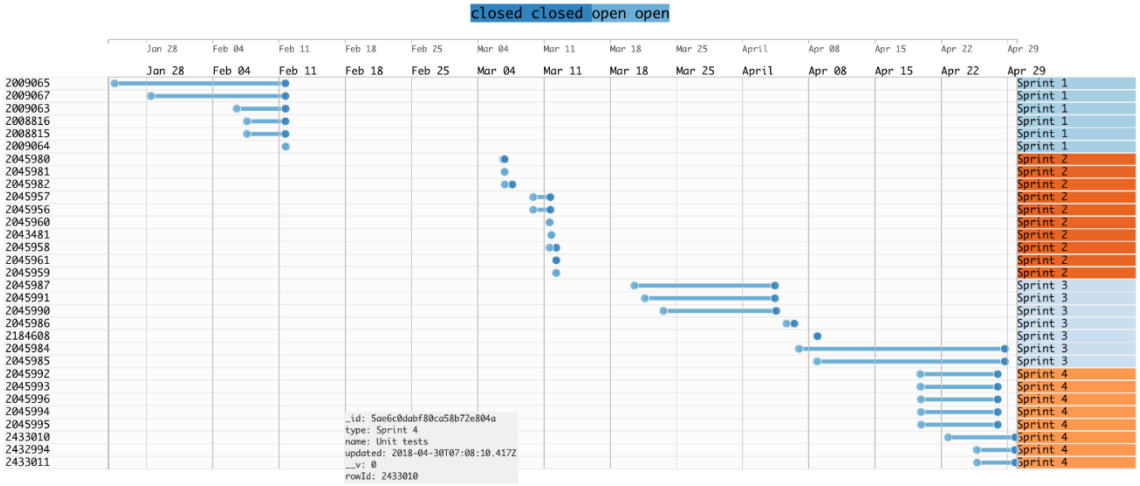
06: A story that has been opened quite late in sprint 2, has not been closed in sprint 2 but only in sprint 3. Two stories of sprint 4 have been opened already in sprint 3. Otherwise pretty structured system.



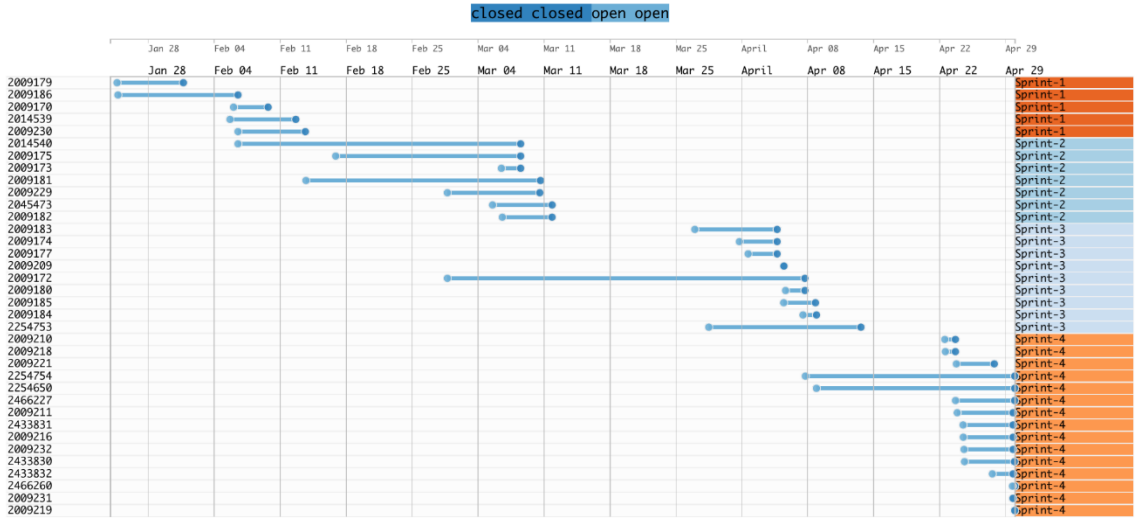
07 ???



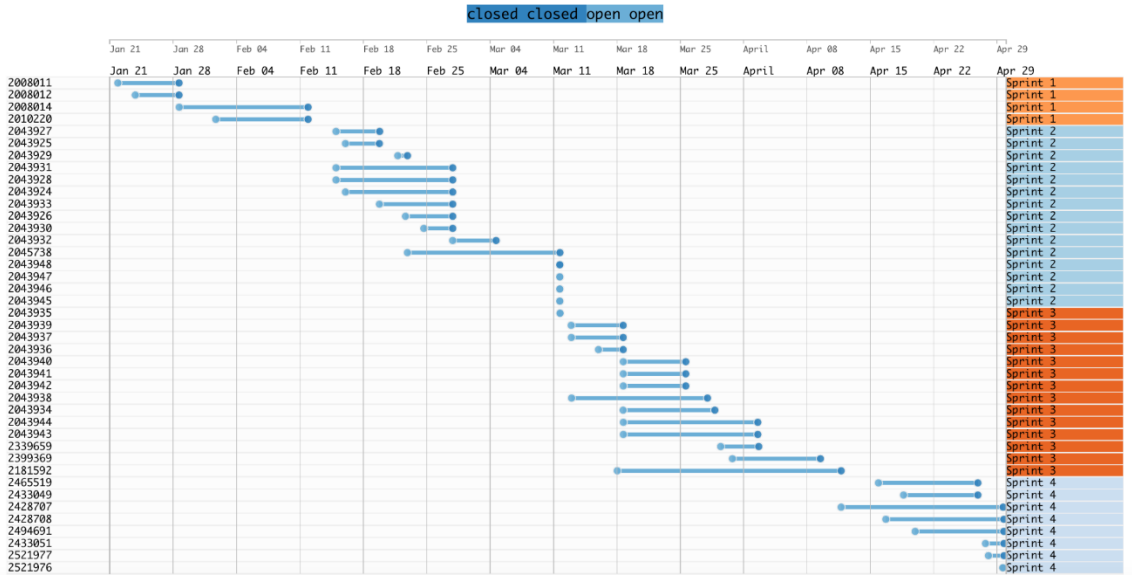
14 Somehow the group has not used Agilefant well in sprint 2 or then the work they have assigned there has been really trivial. Sprint 3 seems to have a few stories actually intended to sprint 4.



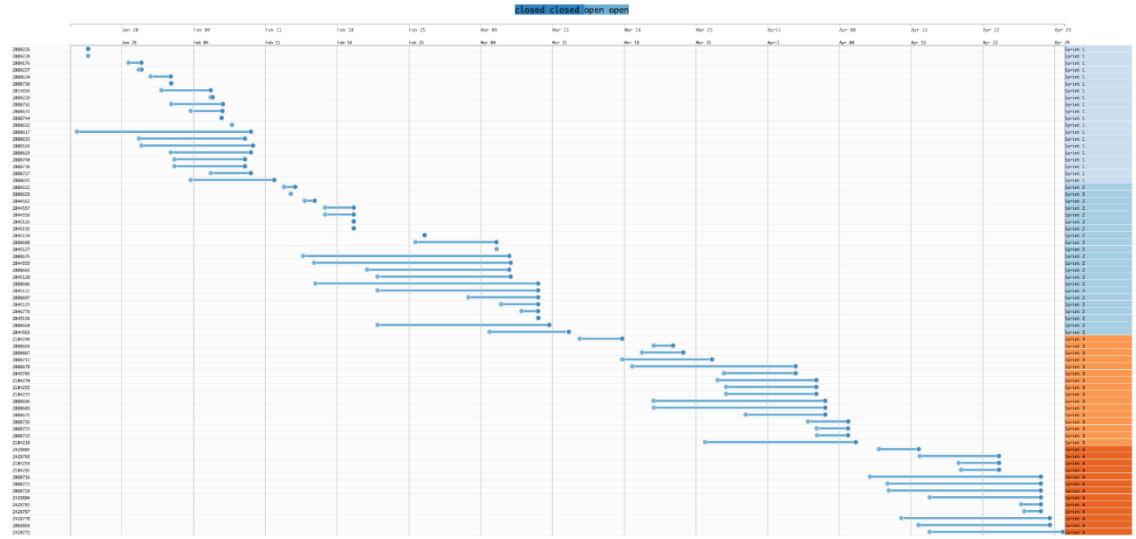
15 Each sprint has some stories closing that have been opened already in the previous sprint.



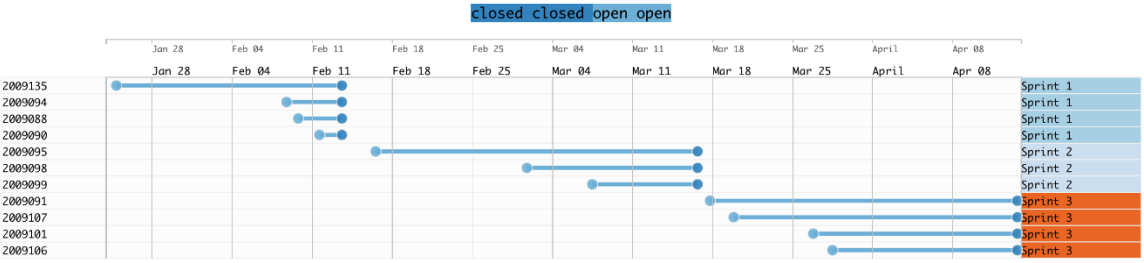
16 Other ways well structured process, but sprint 2 has some “orphan” stories that have not been closed at all. I wonder what happened there?



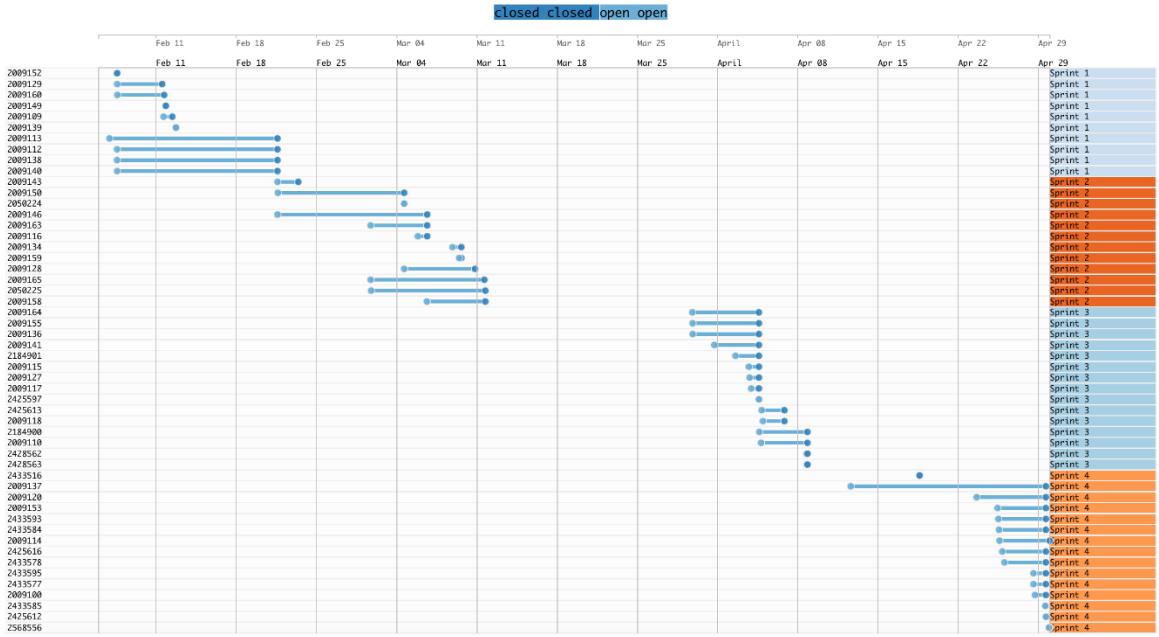
17 The sprints are reflected well in the visualization.



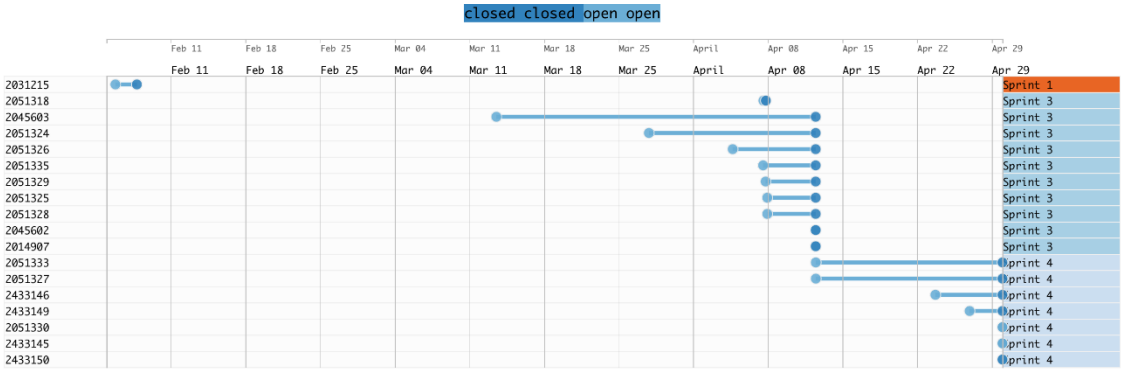
18 Clear closing points for stories. The stories are probably not divided well if there’s only one big story for each developer per sprint. And where’s sprint 4?



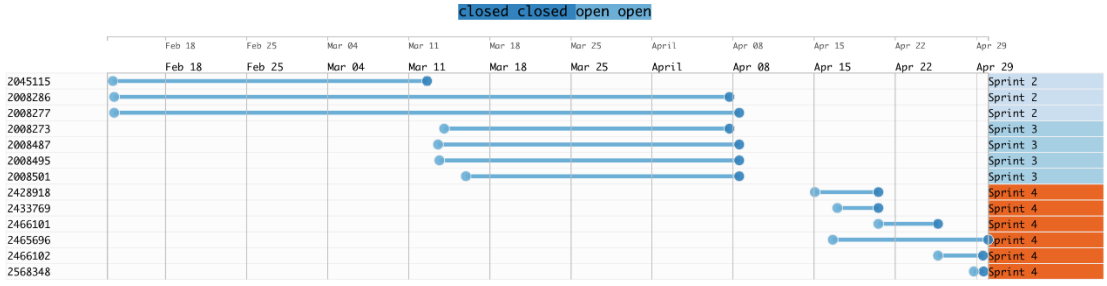
19 The process for closing stories has changed during the project (a change in definition of done?) Clear vacation time (or something similar in March). Good work estimation throughout the project because the amount of stories per sprint has remained very similar.



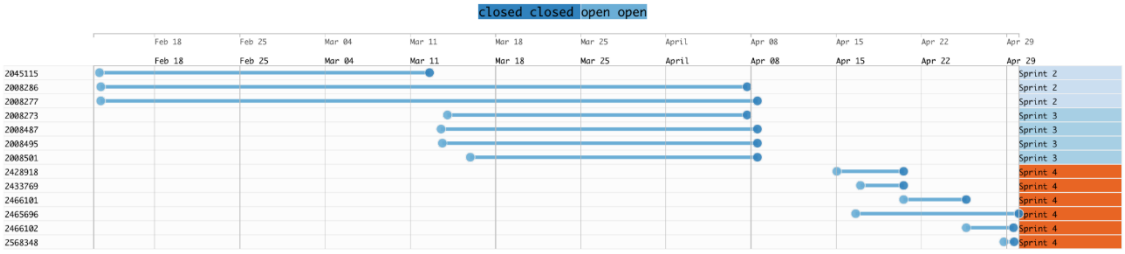
20 No use of Agilefant in half of the project. Sprint 3 review session and sprint 4 planning probably done at the same time.



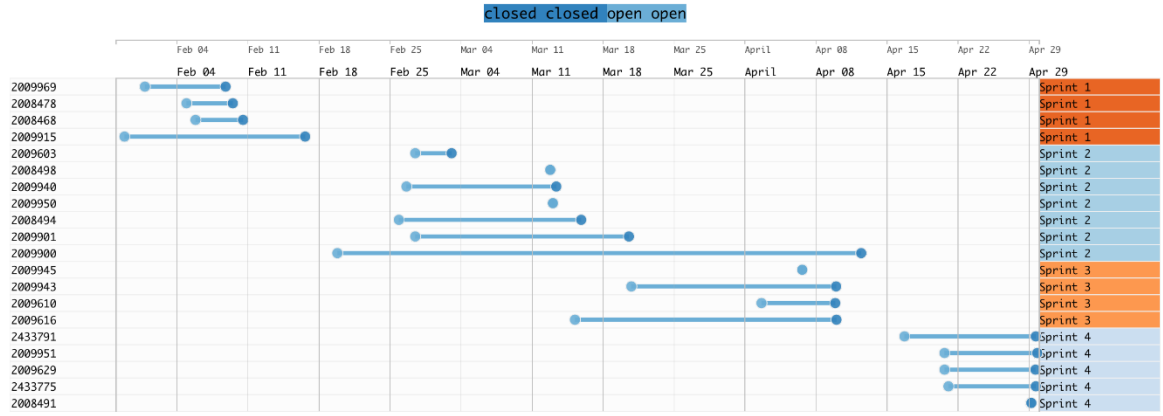
21 No use of agilefant in sprint 1 and possibly not much work done in sprint 2 (because majority of open stories continue to sprint 3. Maybe only 1-2 developers on the job?



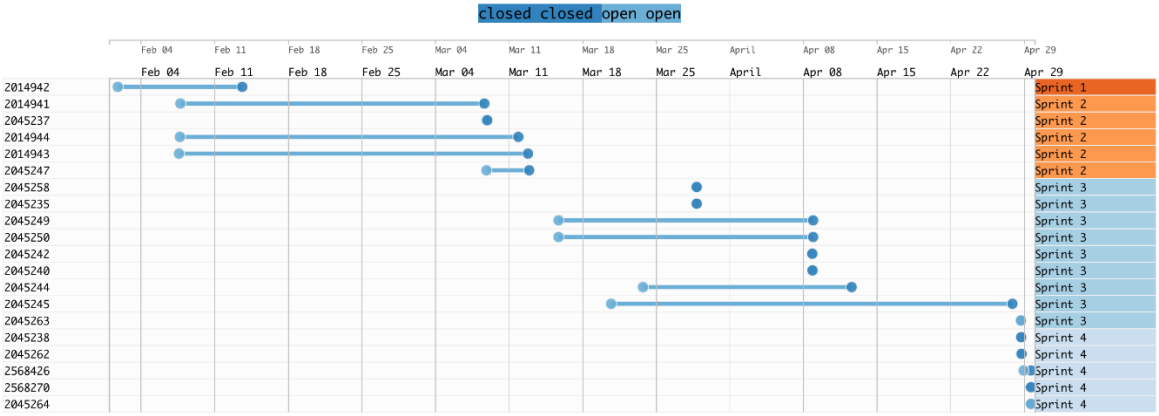
22 Same as 21.



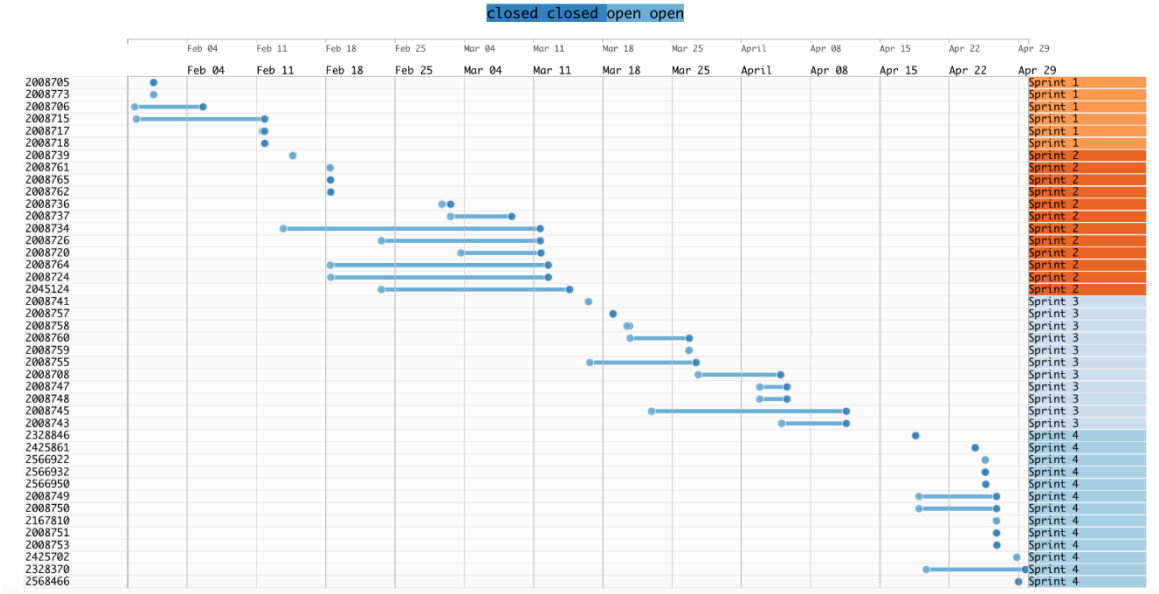
23 Sprints 2 and 3 somehow are intertwined. The work in sprints 1 and 4 seems to be better estimated and worked on.



24 No (proper) use of agilefant in sprints 1 and 4. Two stories in sprint 3 are strange in that the first one has been worked on probably also in sprint 4 and the other has been opened in the last few days of the project and then just left open.



25 Either the group has not used Agilefant too well or then they haven't estimated the amount of work too well. They have a lot of stories that have been opened and closed on the same day. Sprints are reflected well though.



27 Maybe the group has worked on two stories in sprint 2 at first, and then not made new stories but rather opened the same stories again in sprint 4. Some continuation of work from sprint 3 to sprint 4 as well. No use of Agilefant (or only trivial work done) in sprint 1.